

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re U.S. Patent Application)

Applicant: Makoto Nakanishi)

Serial No.)

Filed: October 24, 2003)

For: PARALLEL PROCESSING)
METHOD FOR INVERSE)
MATRIX FOR SHARED)
MEMORY TYPE SCALAR)
PARALLEL COMPUTER)

I hereby certify that this paper is being deposited with the United States Postal Service as EXPRESS MAIL in an envelope addressed to: Mail Stop PATENT APPLICATION, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on this date.

Oct. 24, 2003
Date


Express Mail Label No.: EV032735309US

CLAIM FOR PRIORITY

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Applicant claims foreign priority benefits under 35 U.S.C. § 119 on the basis of Japanese Patent Application No. 2003-074548, filed March 18, 2003. A certified copy of the priority document is enclosed.

In addition, a certified copy of Japanese Patent Application No. 2002-079909, filed March 22, 2002 in Japan, was filed in Application Serial No. 10/288,984 on November 6, 2002.

Respectfully submitted,
GREER, BURNS & CRAIN, LTD.

By



Patrick G. Burns
Registration No. 29,367

October 24, 2003
300 South Wacker Drive
Suite 2500
Chicago, Illinois 60606
Telephone: 312.360.008
Facsimile: 312.360.9315

JAPAN PATENT OFFICE

This is to certify that the annexed is a true copy of the following application as filed with this office.

Date of Application: March 18, 2003

Application Number: Patent Application No. 2003-074548
[ST.10/C] [JP2003-074548]

Applicant(s): FUJITSU LIMITED

August 18, 2003

Commissioner,
Japan Patent Office Yasuo IMAI

Certificate No. P2003-3067303

1503.68591
312,360,0000

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日 2 0 0 3 年 3 月 1 8 日
Date of Application:

出 願 番 号 特 願 2 0 0 3 - 0 7 4 5 4 8
Application Number:
[ST. 10/C]: [J P 2 0 0 3 - 0 7 4 5 4 8]

出 願 人 富士通株式会社
Applicant(s):

2 0 0 3 年 8 月 1 8 日

特許庁長官
Commissioner,
Japan Patent Office

今 井 康 夫



出証番号 出証特 2 0 0 3 - 3 0 6 7 3 0 3

【書類名】 特許願

【整理番号】 0253797

【提出日】 平成15年 3月18日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 17/16

【発明の名称】 共有メモリ型スカラ並列計算機用逆行列の並列処理方法

【請求項の数】 5

【発明者】

【住所又は居所】 神奈川県川崎市中原区上小田中 4 丁目 1 番 1 号 富士通株式会社内

【氏名】 中西 誠

【特許出願人】

【識別番号】 000005223

【氏名又は名称】 富士通株式会社

【代理人】

【識別番号】 100074099

【住所又は居所】 東京都千代田区二番町 8 番地 2 0 二番町ビル 3 F

【弁理士】

【氏名又は名称】 大菅 義之

【電話番号】 03-3238-0031

【選任した代理人】

【識別番号】 100067987

【住所又は居所】 神奈川県横浜市鶴見区北寺尾 7 - 2 5 - 2 8 - 5 0 3

【弁理士】

【氏名又は名称】 久木元 彰

【電話番号】 045-573-3683

【先の出願に基づく優先権主張】

【出願番号】 特願2002- 79909

【出願日】 平成14年 3月22日

【手数料の表示】

【予納台帳番号】 012542

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9705047

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 共有メモリ型スカラ並列計算機用逆行列の並列処理方法

【特許請求の範囲】

【請求項 1】 共有メモリ型スカラ並列計算機用逆行列の並列処理方法を実現させるプログラムであって、

逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、
該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右横、右下のそれぞれのブロックに分解するステップと、

該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列に LU 分解するステップと、

左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、

所定の正方形ブロックの更新を複数段に分けて、1つのプロセッサで行うステップと、

該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと、
を備えることを特徴とする方法を共有メモリ型スカラ並列計算機に実現させるプログラム。

【請求項 2】 前記共有メモリ型スカラ並列計算機は、複数のプロセッサと、該プロセッサに対応して設けられる複数のキャッシュメモリと、複数の共有メモリと、これらを通信可能なように接続する相互結合網とからなることを特徴とする請求項 1 に記載のプログラム。

【請求項 3】 上記方法は、G a u s s - J o r d a n の方法をブロック毎に並列に計算する構成としたことを特徴とする請求項 1 に記載のプログラム。

【請求項 4】 前記各ブロックを分割して並列計算するさいの分割の幅は、逆行列を求める行列の大きさと並列処理のために利用できるプロセッサ数から、並列処理しない正方形ブロックの演算量の総和が演算全体の 1 % 程度となるように設定されることを特徴とする請求項 1 に記載のプログラム。

【請求項 5】 共有メモリ型スカラ並列計算機用逆行列の並列処理方法であって、

逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、
該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右横、右下のそれぞれのブロックに分解するステップと、

該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列に L U 分解するステップと、

左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、

所定の正方形ブロックの更新を複数段に分けて、1つのプロセッサで行うステップと、

該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと、
を備えることを特徴とする方法。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

本発明は、共有メモリ型スカラ並列計算機における演算処理に関する。

【0 0 0 2】

【従来の技術】

従来、ベクトル計算機では、ある行列の逆行列を求める場合、G a u s s - J o r d a n の方法をベースに、メモリアクセス系の動作が速いことを利用した計算方法を用いて行っていた。例えば、ループ内の命令を複数回分明的に記述する 2 重アンローリング等の手法が用いられていた。

【0 0 0 3】

G a u s s - J o r d a n の方法（あるいは、単に G a u s s 法）による逆行列を求める方法を以下に説明する（なお、以下の説明ではピボットの入れ替えは省略するが実際には、ピボットの入れ替えのため行ベクトルの入れ替えを行って

いる。

まず、 A を逆行列を求めるべき行列、 x 、 y は、適当な列ベクトルとする。

$Ax = y$ は、行列要素をあらわに記載すると以下のような連立一次方程式となる。

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = y_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = y_2$$

.....

$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = y_n$$

上記方程式を、 $By = x$ という形に変形すると、 B が A の逆行列となって、逆行列が求まることになる。

- 1) 1行目の方程式を a_{11} で割る。
- 2) i 行目 ($i > 1$) - 1行目 $\times a_{i1}$ を演算する。
- 3) 2行目の方程式の x_2 の係数を1にするよう、2行目に x_2 の係数の逆数をかける。
- 4) i 行目 ($i > 2$) - 2行目 $\times a_{i2}$ を演算する。
- 5) 以上の演算を $n-1$ 行目まで続ける。

【0004】

ピボットの入れ替えに伴う列ベクトルの入れ替えは以下の通りである。

$Ax = y$ の両辺にピボットの入れ替え処理に対応する行列 P をかける。

$$PAx = Py = z$$

行列 B について、以下が成り立つとすると、

$$x = Bz$$

B は、以下のように与えられる。

$$B = (PA)^{-1} = A^{-1}P^{-1}$$

つまり、求まった B に右から P をかけることにより、 A の逆行列が求まる。実際には、列ベクトルの入れ替えを行う必要がある。

【0005】

なお、ここで、 $P = P_n P_{n-1} \cdots P_1$ であり、 P_n は、行列の要素が $P_{ii} = 0$ 、 $P_{ij} = 1$ 、 $P_{jj} = 0$ 、 $P_{ji} = 1$ となる直交変換である。

【0006】

【発明が解決しようとする課題】

ベクトル計算機においては、メモリアクセス系の動作速度が速いことを前提として、上記のような方法で逆行列の演算を行っていたが、共有メモリ型スカラ計算機の場合、演算する行列が大きくなるほどに、共有メモリにアクセスする回数が多くなり、アクセス速度の遅い共有メモリへのアクセスによって、計算機の性能が大きく損なわれてしまうという問題がある。そこで、共有メモリ型スカラ計算機の各プロセッサに設けられる、アクセス速度の速いキャッシュメモリを有効に使い上記のような行列計算を行う必要がある。つまり、行列の各行あるいは列毎に演算していると、共有メモリへのアクセスが多くなってしまうので、行列をブロック化して、各プロセッサにキャッシュメモリに格納されたデータを最大限処理した後、共有メモリにアクセスするようにして、共有メモリへのアクセス数を減少する、各プロセッサに局所化したアルゴリズムが必要となる。

【0007】

本発明の課題は、共有メモリ型スカラ計算機において、高速に逆行列の演算が行える演算の並列処理方法を提供することである。

【0008】

【課題を解決するための手段】

本発明の方法は、共有メモリ型スカラ並列計算機用逆行列の並列処理方法であって、逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右横、右下のそれぞれのブロックに分解するステップと、該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列にLU分解するステップと、左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、所定の正方形ブロックの更新を複数段に分けて、1つのプロセッサで行うステップと、該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと

を備えることを特徴とする。

【0009】

本発明によれば、逆行列の演算をブロック毎の更新処理とし、各ブロックの更新を複数のプロセッサで並列に行うことにより、プロセッサに対応して設けられるキャッシュに格納されたブロックに対し、最大限演算を行った後に、共有メモリへのアクセスをするようにしているので、アルゴリズムが局所化し、高速な逆行列を求めるための演算が行える。

【0010】

【発明の実施の形態】

本発明の実施形態では、ある行列の逆行列を求めるアルゴリズムを提供する。共有メモリ型スカラ並列計算機では、ロードに対する演算量を増やす必要がある。このため、本発明の実施形態では、行列積の形で演算を効率的に行うブロック化した方法を提供する。更に、行列積を使う更新で必要となる行列部分を大きさの変化する行列積を利用する再帰プログラムで記述することで演算密度を高める。

【0011】

図1は、本発明の実施形態が前提とする共有メモリ型スカラ計算機のハードウェア構成を示した図である。

プロセッサ10-1～10-nは、1次キャッシュメモリを持っており、この1次キャッシュメモリはプロセッサの中に組み込まれていることもある。また、各プロセッサ10-1～10-nには、2次キャッシュメモリ13-1～13-nが設けられ、2次キャッシュメモリ13-1～13-nが相互結合網12に結合されている。また、相互結合網12には、共有メモリであるメモリモジュール11-1～11-nが設けられ、プロセッサ10-1～10-nは、演算に必要なデータをここから読み出し、相互結合網12を介して、2次キャッシュメモリ13-1～13-nあるいは、1次キャッシュメモリに記憶させて、演算を行う。

【0012】

この場合、メモリモジュール11-1～11-nから2次キャッシュメモリ1

3-1 ~ 13-n あるいは 1 次キャッシュメモリにデータを読み込んだり、2 次キャッシュメモリ 13-1 ~ 13-n、あるいは、1 次キャッシュメモリから演算後のデータをメモリモジュール 11-1 ~ 11-n に書き込むのはプロセッサ 10-1 ~ 10-n の演算速度に比べて非常に遅い。従って、このような書き込み、読み出しが頻繁に発生すると計算機全体の性能を劣化させてしまう。

【0013】

従って、計算機全体の性能を高く維持するためには、メモリモジュール 11-1 ~ 11-n へのアクセスをできるだけ減らし、2 次キャッシュメモリ 13-1 ~ 13-n、1 次キャッシュメモリ、及びプロセッサ 10-1 ~ 10-n からなるローカルな系においてできるだけ多くの演算をしてしまうようなアルゴリズムが必要となる。

【0014】

したがって、本願発明の実施形態においては、以下のようにして逆行列の計算を行う。

演算すべき行列内のあるブロック幅のブロックに関して右方向の更新のみを行う。こうすることで、ブロックの外での更新で使う情報を残すようにする。つまり、消去の過程で本来機軸として選んだ列ベクトルと直交する行ベクトルの外積で残りの部分を更新することが従来の技術で説明した Gauss-Jordan の方法において行われることが分かる。すなわち、前述の Gauss-Jordan の方法の 2) の方法をあらわに書くと、

$a_{ij} - c_{1j} \times a_{i1}$ となっている ($i, j > 1 : c_{1j} = a_{1j} / a_{11}$)。

【0015】

この式の意味するところは、1 列目の列ベクトルを機軸として選んで、これと直交する 1 行目の行ベクトルを選択すると、これらを除いた 2 行目以降 \times 2 列目以降からなる行列は、1 列目の列ベクトルと 1 行目の行ベクトルの外積で更新されるということである。

【0016】

また、 $Ax = y$ を I を単位行列として書き直すと、 $Ax = Iy$ となり、 A を Gauss-Jordan 法によって変換した場合、右辺の単位行列も変換を受け

る。

【0017】

この単位行列の更新においては、Aにおいて消去された列に対応する列の（ i 、 i ）要素は、右辺の行列においては $1/a_{ii}$ となり、右辺の行列の他の要素は、Aの消去された列ベクトルの要素に $1/a_{ii}$ をかけて -1 を乗じた数となる。また、列ブロック行列でも左側の消去が必要になる。

【0018】

図2～図8は、本発明の実施形態の計算順序を説明する図である。

計算の順序は、次のようになる。

まず、図2において、行列Mを列ブロック行列毎に左からブロック化して、上で述べた方法で更新を行う。

- 1) EとHをLU分解する（特願平12-358232号参照）。
- 2) BをEの上三角部分Uを利用して $B \leftarrow B U^{-1}$ と更新する。
- 3) Eの下三角部分を利用して $D \leftarrow L^{-1} D$ 及び $F \leftarrow L^{-1} D$
- 4) A、C、G、Iを更新する。

【0019】

$$A \leftarrow A - B \times D, C \leftarrow C - B \times F, G \leftarrow G - H \times D, I \leftarrow I - H \times F$$

- 5) Eの上三角部分の更新を行う。この部分はLU分解では未更新となっているため、このタイミングで更新する。
- 6) 次にD及びFの上部に対する更新をD、Fの情報とEの情報を使って更新する。

【0020】

このとき、演算密度を高めるために再帰的なプログラムで行列積を使って更新する。

a) 再帰的プログラム

D及びFの上部の更新は、各々の行ベクトルとEのそれと直交する列ベクトルの外積で計算できる。演算密度を上げるために次のような再帰的プログラムで行う。ここで、D及びFの上部とは、図8で説明するように、D及びFの行ブロックの上の方と言う意味であり、図8から明らかなように、D及びFの更新は、こ

れら行ブロックの上部から行われる。

【0021】

```
recursive subroutine rowupdate ()
```

```
if (更新幅 < 10) then
```

Eの上三角行列から対角要素を除いた行列と行ブロック行列をかけて差し引く
(詳細は、図8の説明参照)

```
else
```

c 更新幅を前半と後半に分ける

前半部分を更新する

c 次に後半部分をrowupdateを呼び出して更新する

```
call rowupdate ()
```

```
return
```

```
end
```

7) 次に対角線上の要素を右下に移動させながら、この対角の左下の列方向の長方形部分を更新する。この結果、B及びHの更新に必要な情報を作り出す。

8) B、Hに関して左側の更新を行う。これは以下のように行う。

【0022】

ブロック幅をdとして、左からi、・・・、 $i + d - 1$ と番号を振る。

a_{ii} を逆数にする。他の部分は a_{ji} で割って、符号を逆にする。

$i + 1$ 行の左を $a_{i+1, i+1}$ で割り、 $a_{i+1, i+1}$ を逆数にする。

【0023】

左部分の更新を左部分の $i + 1$ 行と $i + 1$ 列をかけて引くことで行う。

これを繰り返す。

この部分は、更に以下の手順に分かれる。これらも再帰的なアルゴリズムに組み立て直すことが必要となる。

9) 最後にEの対角線上の要素から見て左上部分の行方向の長方形部分を更新する。

10) 最後に列ベクトルをピボットの入れ替えの履歴の逆方向に入れ替える処理を行う。

5) で更新する部分は図3の斜線部分(E2)を、対角要素を対角線にスライドさせながら

$E2 = E2 - a \times b$ で更新する。

【0024】

この部分はLU分解の処理では未更新である。これを行うことで、D、Fの更新で必要になる上三角行列の情報が得られる。

7) で更新する部分は図4の斜線部分を対角要素をスライドさせながら、 $E3 = E3 - c \times d$ で更新する。

【0025】

この更新に先立ち、cは対角要素の逆数をかける。

更新の後の対角要素は元の対角要素の逆数とする。dは、各列要素に更新前の対角要素をかけて、-1をかけたものとする。

【0026】

この結果、B及びHの左方向の更新に必要な下三角行列の情報が得られる。

9) で、図5の斜線の部分を対角要素をスライドさせながら、

$E1 = E1 - a \times c$ で更新する。

【0027】

更新の後のaは、aの列要素に元の対角要素をかけて、-1をかけたものとする。

D) 共有メモリ型スカラ並列計算機向けの並列化の詳細

0) E、F、H及びIのLU分解はLU分解の並列化アルゴリズムを利用して並列分解する。

1) ブロック幅の制御

本発明の実施形態では、問題規模及び並列化で使うプロセッサ数に応じて、ブロック幅を調節する機構を備える。

【0028】

ブロック E の更新は一つのプロセッサで行うので、この部分のコストが全体のコストから考えて無視できる程度（1 % 程度）になるようにブロック幅を決める。

2) C の 4) の更新はおのこの行列積による更新を並列に行う。各プロセッサで 2 次元目を均等に分割して計算を並列分担する。

3) C の 5) での更新、つまり D 及び F の更新は E の上三角部分を共通に参照して、D 及び F の 2 次元目を均等に分割した領域を各プロセッサが並列に更新する。

4) C の 8) の更新は及び H の 1 次元目を均等に分割した領域を各プロセッサは並列分担して更新する。

5) 最後に、列ベクトルの入れ替え処理は、行列全体の 1 次元目を均等に分割して、各領域の部分に関する入れ替えを並列に行う。

【 0 0 2 9 】

図 6 の破線は各行列部分を並列に計算する更新を並列に行う領域の分割例を示している。

・ B または H の更新についてのメモリのアクセス
深さ 2 まで再帰プログラムが動作した場合を解説する。

【 0 0 3 0 】

図 7 (a) と (b)、(c) と (d)、(e) と (f) が組になっている。

最初に、図 7 (a) の斜線部分を更新する。このとき斜線部分及び図 7 (b) の破線三角部分を使う。

【 0 0 3 1 】

詳細は、疑似コード参照。次に、図 7 (a) 斜線部分を図 7 (a) 横線部分と図 7 (b) 斜線四角部分を使って更新する。その後、図 7 (a) 横線部分を図 7 (b) 太線三角部分を使って更新する。

【 0 0 3 2 】

次に、図 7 (c) 斜線部分を、図 7 (c) 右白塗り四角と図 7 (d) 太線四角の積で更新する。

その後、図 7 (e) 斜線部分を図 7 (f) 破線三角部分を使い更新する。更に、図 7 (e) 斜線部分を図 7 (e) 横線部分と図 7 (f) 斜線四角の積で更新する。最後に横線部分を図 7 (f) 太線三角部分を使って更新する。

【0033】

上記手順において、E の参照は共通である。従って、E は各プロセッサのキャッシュに格納して、参照するようにしても良い。また、例えば、B の参照・更新は行方向（太線破線で分割）した領域に関して並列に処理する。

【0034】

・ D または F の更新におけるメモリアクセス

深さ 2 まで再帰プログラムが動作した場合を解説する。

【0035】

最初に、図 8 (a) 左の斜線部分を更新する。このとき斜線部分及び図 8 (a) 右の破線三角部分を使う。

詳細は、疑似コード参照。次に、図 8 (a) 左の斜線部分を図 8 (a) 右の斜線四角部分と図 8 (a) 左の縦線部分を使って更新する。その後、図 8 (a) 左の横線部分を図 8 (a) 右の太線三角分を使って更新する。

次に、図 8 (b) 左の斜線部分を、図 8 (b) 右の太線四角と図 8 (b) 左の白塗り四角の積で更新する。

その後、図 8 (c) 左の斜線部分を図 8 (c) 右の破線三角部分を使い更新する。更に、図 8 (c) 左の斜線部分を図 8 (c) 右の斜線四角と図 8 (c) 左の縦線部分との積で更新する。最後に図 8 (c) 左の縦線部分を図 8 (c) 右の太線三角部分を使って更新。

【0036】

上記手順において、E 参照は共通である。従って、各プロセッサのキャッシュに E を格納して参照するようにしても良い。また、D の参照・更新は行方向（太線破線で分割）した領域に関して並列に処理する。

【0037】

図 9 ～図 15 は、本発明の実施形態の疑似コードである。

図 9 は、逆行列の並列化アルゴリズムのメインアルゴリズムの疑似コードである。以下の疑似コードで、左端に c と付いている行はコメント行である。array a(k,n) は、逆行列を求めるべき行列の要素を格納する配列である。ip (n) は、LU 分解のサブルーチンにおいて、行の入れ替えを行った際の情報が格納される配列である。LU 分解のサブルーチンのアルゴリズムは、特願平 1 2 - 3 5 8 2 3 2 号を参照されたい。nb は、LU 分解する際に指定するブロックの数を示す。

【 0 0 3 8 】

1 つの指定ブロックについての LU 分解が終わると、ip (i) が i より大きいとき、行列の i 行目を ip (i) 行目と入れ替える。そして、update サブルーチンを呼び出して、行列の更新を行う。LU 分解から update サブルーチンまでの処理は、指定されるブロック全てについて処理し終わるまで繰り返す。そして、最後の指定ブロックについては、別途 LU 分解と update をおこなって処理を終了する。

【 0 0 3 9 】

図 1 0 は、ブロック LU 分解の情報を利用して残りの部分の更新を行うルーチンの疑似コードである。

同図の更新ルーチンでは、ブロック A ~ H に従って、それぞれの更新を行う。ブロック A ~ D 及び G については、専用のサブルーチンを更に読み出す。ブロック I は、LU 分解の際に既に更新されているので、ここでは別途サブルーチンを設けることはしない。

【 0 0 4 0 】

ブロック A ~ D 及び G の更新ルーチンが終了した後、バリア同期を取る。その次に、プロセッサの番号（スレッドの番号）が 1 の場合に、「e の更新 1」を行う。これは、e - update 1 サブルーチンによって行う。その後、バリア同期を取る。

【 0 0 4 1 】

len は、1 つのスレッドが処理するブロックの幅である。is は、処理するブロックの最初の位置であり、ie は、処理するブロックの最後の位置である。

`df-update` は、ブロック D 及び F を更新するサブルーチンである。ブロック D、F の更新が終わると、ブロックの最初の位置にブロック幅を加えたものを新たなブロックの最初の位置 (`nbase2`) として格納し、`len` を新たに計算し、ブロックの最初と最後の位置 `is2` と `ie2` を新たに計算し、`df-update` で D と F の更新を行い、バリア同期を取る。

【0042】

更に、「e の更新 2」として、スレッドの番号が 1 の場合に、ブロック E の更新サブルーチン `e-update2` を呼び出し、バリア同期を取る。上記と同様に、`len`、`is`、`ie` を計算し、ブロック B と H の更新ルーチン `bh-update` を呼び出し、その後、`nbase2` を求めて、`len`、`is2`、`ie2` を求め、再び `bh-update` で処理を行い、バリア同期を取る。

【0043】

更に、スレッドの番号が 1 の時、「e の更新 3」として、`e-update3` で処理を行い、処理後バリア同期を取る。

更に、その後、ピボットの入れ替えたままの状態を元に戻すため、`len`、`is`、`ie` を計算した後、サブルーチン `exchange` によって列の入れ替えを行って、バリア同期を取り、スレッドを消去して、処理を終了する。

【0044】

図 11 は、ブロック B とブロック D の更新サブルーチンの疑似コードである。

ブロック B の更新においては、サブルーチン `b-update` は、共有の行列配列 $a(k, n)$ にアクセスし、上記説明と同じ意味を持つ `len`、`is1`、`ie1` を計算する。`iof` は、B ブロックの始まりの列の番号である。そして、E ブロックの上三角行列の対角要素を 1 にした行列 $TRU - U$ を使って、同図に示す式によって行列 a のブロック B の更新をする。ここで、 $is : ie$ という記号は、行列要素の is から ie までについて処理をすることを意味する。

【0045】

また、ブロック D の更新においては、サブルーチン `d-update` によって、同様のパラメータを計算し、ブロック E の下三角行列 TRL によって同図の式のように行列 a の更新を行う。

【0046】

図12は、ブロックCとAの更新サブルーチンの疑似コードである。

ブロックCの更新サブルーチン `c-update` では、ブロックBとFの乗算によってブロックCを更新する。 $a(1:iof, is2:ie2)$ がブロックCを表し、 $a(1:iof, iof+1:iof+blk)$ がブロックBを、 $a(iof+1:iof+blk, is2:ie2)$ がブロックFを表す。

【0047】

ブロックAの更新サブルーチン `a-update` では、ブロックBとDでブロックAを更新する。 $a(1:iof, is2:ie2)$ がブロックA、 $a(1:iof, iof+1:iof+blk)$ がブロックB、 $a(iof+1:iof+blk, is2:ie2)$ がブロックDである。

【0048】

図13は、ブロックG、Eの最初と2回目の更新を表す疑似コードである。

ブロックGの更新サブルーチン `g-update` においては、前述のサブルーチンと同様に、ブロックの幅や開始位置、終了位置などを示す `len`、`is2`、`ie2`、`iof`などを計算し、ブロックGをブロックDとHで更新する。 $a(iof+1:n, is2:ie2)$ がブロックGであり、 $a(iof+1:n, iof+1:iof+blk)$ がブロックH、 $a(iof+1:iof+blk, is2:ie2)$ がブロックDである。

【0049】

ブロックEの最初の更新サブルーチン `e-update1` では、Eの対角成分より上の三角行列を対角成分以前の列ベクトル $s(1:i, i)$ と、該対角成分以降の行ベクトル $(i, i+1:blk)$ で更新する。

【0050】

ブロックEの2回目の更新サブルーチン `e-update2` では、ブロックEの上三角行列の対角成分を更新前の要素値を対角成分値で割った値に更新し、対角成分以前の行ベクトル $s(i, 1:i-1)$ と対角成分以後の列ベクトル $s(i+1:blk, i)$ で更新し、ブロックEの下三角行列の要素値を対角成分の符号を変えたもので割った値に更新し、対角要素にブロックEの対角要素の逆数

に更新する。

【0051】

図14は、ブロックEの最後の更新、ブロックD及びFの更新サブルーチンの疑似コードである。

ブロックEの最後の更新サブルーチン `e-update3` では、ブロックEの上三角行列を対角要素以前の列ベクトル $s(1:i-1, i)$ と行ベクトル $s(i, 1:i-1)$ で更新し、ブロックEの対角要素以前の要素に更新前の対角要素をかけて更新する。

【0052】

ブロックD及びFの更新サブルーチン `df-update` においては、ブロックの幅 `len` が10より小さい場合、ブロックDあるいはブロックF（サブルーチンの引数の `is`、`ie` によって決定される）をブロックEの要素値 $s(1:i-1, i)$ と自身の行ベクトル $a(i, is:ie)$ で更新する。ここで、ブロックDあるいはFの要素値が $a(1:i-1, is:ie)$ となっているのは、このサブルーチンが行列要素を読み込む場合、前述の `nbase` によって読み込む位置がオフセットされて読み込まれているため、列番号が $1 \sim i-1$ の要素値について計算することがブロックDあるいはFについて研鑽することになる。`len` が20以上、32以下の場合には、`len1`、`len2` を定義し、`df-update` を再帰的に呼び出し、同図の処理をした後、更に、`df-update` を呼び出して処理を行い終了する。

【0053】

図15は、ブロックB及びHの更新サブルーチンの疑似コードである。

同図においては、`bh-update` は、`len` が10より小さいとき、同図の演算によって更新し、`len` が20以上、32以下の時には、`len1`、`len2` を定義し、その他の場合には、`len1`、`len2` を別に定義し、`bh-update` を呼び出し、同図の式による演算を行い、更に `bh-update` を呼び出し、処理をして終了する。

【0054】

本発明の実施形態によれば、同じ機能（LU分解した後、逆行列を求める別の

方法)で、SUNの数値計算ライブラリSUN Performance libraryの機能に比べて7個のCPUで6.6倍高速となる。

【0055】

図16～図29は、疑似コードの処理をフローチャートで表したものである。

図16は、メインの処理である。まず、逆行列の演算開始として、サブルーチンとしてshared配列A(k, n)を入力する。ステップS10では、スレッドを生成し、各スレッドでローカル域numthrに総スレッド数、nothrldに各スレッドに割り振られたスレッド番号を設定する。また、各スレッドでiblkにブロック幅を設定し、nb=(n+iblk-1)/iblkを設定し、i=1を設定する。ステップS11では、iがnb-1に等しいか否かを判断する。ステップS11の判断がYESの場合には、ステップS17に進む。ステップS11の判断がNOの場合には、ステップS12において、nbaseを(i-1)×iblkを設定し、ステップS13で、A(nbase+1:n, nbase+1:nbase+n)の部分をブロック幅iblkでブロックLU分解し、行ブロック及び右下正方行列部分を更新する。ここで、ip(nbase+1:nbase+iblk)に行の入れ替えの情報が入っている。これはサブルーチンにしておく。この部分の並列アルゴリズムは先に本出願人が出願した特許出願に記載されている。ステップS14においては、サブルーチンexchgrowを呼び出し、A(nbase+1:nbase+iblk, 1:nbase)部分の行ベクトルの入れ替えを行う。すなわち、jをnbase+1からnbase+iblkまで変化させて、ip(j)>jを満たすときA(j,1:nbase)とA(ip(j),1:nbase)を入れ替える。ステップS15では、サブルーチンupdateを呼んで、他のブロックを更新する。ステップS16では、i=i+1として、ステップS11に戻る。ステップS17では、nbase=(nb-1)×iblkを演算し、ステップS18において、A(nbase+1:n, nbase+1:n)の部分に関して、LU分解を行う。ip(nbase+1:n-1)に行の入れ替えの情報が入っている。ステップS19において、サブルーチンexchgrowを呼び出し、A(nbase+1:n-1, 1:nbase)部分の行ベクトルの入れ替えを行う。ステップS20において、サブルーチンupdateを呼んで更新する。ステップS21では、サブルーチンexhgcolを呼んで列ベクトルを入れ替える。すなわち、jをnから1ずつ減らして変化させて、ip(j)>jを満たすとき、A(1:n, j)、A(1:n, ip(j))を入れ替える。ステップS22では、並列

処理のために生成したスレッドを消去する。

【 0 0 5 6 】

図 1 7 及び図 1 8 は、サブルーチンupdateのフローチャートである。

ステップ S 3 0 においては、サブルーチンb-update、d-update、c-update、a-update、g-updateを呼び出すとき、nbase、iblk、配列 A 及びスレッドの情報であるnumthrd、各スレッドの番号nothrdを受け渡す。サブルーチンb-updateを呼んでブロックを更新する。ステップ S 3 1 では、サブルーチンd-updateを呼んでブロックを更新する。ステップ S 3 2 において、サブルーチンc-updateを呼んでブロックを更新する。ステップ S 3 3 では、サブルーチンa-updateを呼んでブロックを更新する。ステップ S 3 4 では、サブルーチンg-updateを呼んでブロックを更新する。

【 0 0 5 7 】

ステップ S 3 5 においては、各スレッド間でバリア同期を取る。ステップ S 3 6 においては、nothrdの値が 1 のスレッドか否かを判断する。ステップ S 3 6 の判断が N O の場合には、ステップ S 3 8 に進む。ステップ S 3 6 の判断が Y E S の場合には、ステップ S 3 7 において、サブルーチンe-updateを呼んでブロック e を更新し、ステップ S 3 8 に進む。ステップ S 3 8 では、各スレッド間でバリア同期を取る。ステップ S 3 9 においては、df-updateで各スレッドで分担する始点 (is)、終点 (ie) を決めて、受け渡す。 $len=(nbase+numthrd-1)/numthrd$ 、 $is=(nothrd-1) \times len+1$ 、 $ie=\min(nbase, nothrd \times len)$ を演算する。ブロックの 1 次元目先頭istart=nbase+1、ブロック幅len=iblkとする。ステップ S 4 0 においては、サブルーチンdf-updateを呼んでブロック f を更新する。

【 0 0 5 8 】

ステップ S 4 1 においては、同じく始点、終点を計算し受け渡す。 $nbase2=nbase+iblk$ 、 $len=(n-nbase2+numthrd-1)/numthrd$ 、 $is2=nbase2+(nothrd-1) \times len+1$ 、 $ie2=\min(n, nbase2+nothrd \times len)$ を演算する。ステップ S 4 2 においては、ブロックの 1 次元目先頭istart=nbase+1、ブロック幅len=iblkとする。サブルーチンdf-updateを呼んで、ブロック f を更新する。ステップ S 4 3 では、各スレッド間でバリア同期をとる。ステップ S 4 4 nothrdの値が 1 のスレッドか否かを判断

する。ステップ S 4 4 の判断が N O の場合には、ステップ S 4 6 に進む。ステップ S 4 4 の判断が Y E S の場合には、ステップ S 4 5 において、サブルーチン e-update2 を呼んでブロック e を更新し、ステップ S 4 6 に進む。ステップ S 4 6 においては、各スレッド間でバリア同期を取る。ステップ S 4 7 においては、bh-update の各スレッドに受け渡す始点、終点を計算する。すなわち、 $len = (nbase + numthrd - 1) / numthrd$ 、 $is = (nothrd - 1) \times len + 1$ 、 $ie = \min(nbase, nothrd \times len)$ を計算する。ブロックの 1 次元目の先頭 $istart = nbase + 1$ 、ブロック幅 $len = iblk$ とする。サブルーチン bh-update を呼んで、ブロック b を更新する。ステップ S 4 8 においては、bh-update の各スレッドに受け渡す始点、終点を計算する。

【0059】

ステップ S 4 9 においては、 $nbase2 = nbase + iblk$ 、 $len = (n - nbase2 + numthrd - 1) / numthrd$ 、 $is2 = nbase2 + (nothrd - 1) \times len + 1$ 、 $ie2 = \min(n, nbase2 + nothrd \times nothrd \times len)$ を計算し、ブロックの 1 次元目の先頭 $istart = nbase + 1$ 、ブロック幅 $len = iblk$ とする。ステップ S 5 0 においては、bh-update の各スレッドに受け渡す始点、終点を計算する。ステップ S 5 1 においては、各スレッド間でバリア同期をとる。ステップ S 5 2 では、nothrd の値が 1 のスレッドか否かを判断する。ステップ S 5 2 の判断が N O の場合には、ステップ S 5 4 に進み、判断が Y E S の場合には、ステップ S 5 3 において、サブルーチン e-update3 を呼んで、ブロック e を更新し、ステップ S 5 4 に進む。ステップ S 5 4 においては、各スレッド間でバリア同期を取る。

【0060】

図 1 9 は、サブルーチン b-update と d-update の処理を示すフローチャートである。

サブルーチン b-update では、ステップ S 6 0 において、各スレッドで分担する 1 次元目の始点、終点を計算する。すなわち、 $len = (nbase + numthrd - 1) / numthrd$ 、 $isl = (nothrd - 1) \times len + 1$ 、 $iel = \min(nbase, nothrd \times len)$ 、 $iof = nbase$ を計算する。ステップ S 6 1 においては、 $A(isl:iel, iof+1:iof+iblk) = A(isl:iel, iof+1:iof+iblk) \times TRU - U(A(iof+1:iof+iblk, iof+1:iof+iblk))^{-1}$ を計算する。TRU-U は対角要素を 1.0 とした上三角行列部分である。そして、サブルーチ

ンを抜ける。

【 0 0 6 1 】

サブルーチンd-updateでは、ステップS 6 5において、各スレッドで分担する始点、終点を計算する。すなわち、 $len = (nbase + numthrd - 1) / numthrd$ 、 $isl = (nothrd - 1) \times len + 1$ 、 $iel = \min(nbase, nothrd \times len)$ 、 $iof = nbase$ を計算する。ステップS 6 6では、 $A(iof+1:iof+iblk, isl:iel) = TRL(A(iof+1:iof+iblk, iof+1:iof+iblk))^{-1} \times A(iof+1:iof+iblk, isl:iel)$ を計算する。TRLは正方行列の下三角行列部分である。そして、サブルーチンを抜ける。

【 0 0 6 2 】

図 2 0 は、サブルーチンc-updateのフローチャートである。

ステップS 7 0において、各スレッドで分担する2次元目の始点、終点を計算する。すなわち、 $nbase2 = nbase + iblk$ 、 $len = (n - nbase2 + numthrd - 1) / numthrd$ 、 $is2 = nbase2 + (nothrd - 1) \times len + 1$ 、 $ie2 = \min(n, nbase2 + nothrd \times len)$ 、 $iof = nbase$ を計算する。ステップS 7 1では、 $A(1:iof, isl:ie2) = A(1:iof, iof+1:iof+iblk) \times A(iof+1:iof+iblk, is2:ie2)$ を計算し、サブルーチンを抜ける。

【 0 0 6 3 】

図 2 1 は、サブルーチンa-updateのフローチャートである。

ステップS 7 5において、各スレッドで分担する2次元目の始点、終点を計算する。すなわち、 $len = (nbase + numthrd - 1) / numthrd$ 、 $is2 = (nothrd - 1) \times len + 1$ 、 $ie2 = \min(nbase, nothrd \times len)$ 、 $iof = nbase$ を計算する。ステップS 7 6においては、 $A(1:iof, is2:ie2) = A(1:iof, is2:ie2) - A(1:iof, iof+1:iof+iblk) \times A(iof+1:iof+iblk, is2:ie2)$ を計算して、サブルーチンを抜ける。

【 0 0 6 4 】

図 2 2 は、サブルーチンg-updateのフローチャートである。

ステップS 8 0においては、各スレッドで分担する2次元目の始点、終点を計算する。すなわち、 $len = (nbase + numthrd - 1) / umthrd$ 、 $is2 = (nothrd - 1) \times len + 1$ 、 $ie2 = \min(nbase, nothrd \times len)$ 、 $iof = nbase$ を計算する。ステップS 8 1においては、 $A(iof+1:n, is2:ie2) = A(iof+1:n, is2:ie2) - A(iof+1:n, iof+1:iof+iblk) \times A(iof+1:iof+iblk, is2:ie2)$ を計算し、サブルーチンを抜ける。

【0065】

図23は、サブルーチンe-updateのフローチャートである。

このルーチンでは、ブロックEの左上隅の要素がS(k,*)の最初の要素となるように引数で受け取る。ステップS85では、i=1を設定する。ステップS86では、i>iblkか否かを判断する。ステップS86の判断がYESの場合には、サブルーチンを抜ける。ステップS86の判断がNOの場合には、ステップS87において、 $S(1:i-1, i+1:iblk) = S(1:i-1, i+1:iblk) - S(1:i-1, i) \times S(i, i+1:iblk)$ 、i=i+1を計算し、サブルーチンを抜ける。

【0066】

図24は、サブルーチンe2-updateのフローチャートである。

まず、ブロックEの左上隅の要素がS(k,*)の最初の要素となるように引数で受ける。ステップS90では、i=1と設定し、ステップS91において、i>iblkか否かを判断する。ステップS91の判断がYESの場合には、サブルーチンを抜ける。ステップS91の判断がNOの場合には、ステップS92において、 $tmp = 1.0/S(i, i)$ 、 $S(i, i:i-1) = tmp \times S(i, 1:i-1)$ 、 $S(i+1:iblk, 1:i-1) = S(i+1:iblk, 1:i-1) - S(i, 1:i-1) \times S(i+1:iblk, i)$ 、 $S(i+1:iblk, i) = -A(i+1, iblk, i) \times tmp$ 、 $A(i, i) = tmp$ 、i=i+1を演算して、ステップS91に戻る。

【0067】

図25は、サブルーチンe3-updateのフローチャートである。

まず、ブロックEの左上隅の要素がS(k,*)の最初の要素となるように引数で受ける。ステップS95では、i=1と設定し、ステップS96で、i>iblkか否かを判断する。ステップS96の判断がYESの場合には、サブルーチンを抜ける。ステップS96の判断がNOの場合には、ステップS97において、 $S(1:i-1, 1:i-1) = S(1:i-1, 1:i-1) - S(1:i-1, i) \times S(i, 1:i-1)$ 、 $S(1:i-1, i) = S(1:i-1) \times S(i, i)$ 、i=i+1を演算し、ステップS96に戻る。

【0068】

図26は、サブルーチンdf-updateのフローチャートである。

このサブルーチンは、再帰的プログラムとなっている。まず、各スレッドで処理する範囲を示す始点、終点をis、ieで受ける。ブロックEの左上隅の要素がS

(k、*)の最初の要素となるように引数で受ける。処理するブロックの1次元目の先頭istartとブロック幅lenで受ける。ステップS100では、 $len < 10$ か否かを判断する。ステップS100の判断がNOの場合には、ステップS104に進む。ステップS100の判断がYESの場合には、ステップS101において、 $i=1$ と設定し、ステップS102において、 $i \leq len$ か否かを判断する。ステップS102の判断がNOの場合には、サブルーチンを抜ける。ステップS102の判断がYESの場合には、ステップS103において、 $js=istart$ 、 $je=istart-1+len-1$ 、 $A(js:je, is, ie)=A(js:je, is, ie)-S(js:je, i) \times A(istart+i-1, is:ie)$ を演算し、ステップS102に戻る。

【0069】

ステップS104では、 $len \geq 32$ あるいは $len \leq 20$ か否かを判断する。ステップS104の判断がNOの場合には、ステップS106で、 $len1=len/3$ 、 $len2=len-len1$ として、ステップS107に進む。ステップS104の判断がYESの場合には、ステップS105において、 $len1=len/2$ 、 $len2=len-len1$ として、ステップS107に進む。ステップS107では、サブルーチンbf-updateを再帰的に呼び出す（ブロックの先頭のistartとブロック幅のlen1を受け渡す）。ステップS108では、 $js=istart$ 、 $je=istart+len1-1$ 、 $js2=js-nbase$ 、 $je2=js2+len1-1$ 、 $js3=js2+len1$ 、 $je3=js3+len2-1$ 、 $js4=istart+len1$ 、 $je4=js4+len2-1$ 、 $A(js:je, is:ie)=A(js:je, is:ie)-S(js2:je2, js3:je3) \times A(js4, je4:len, is:ie)$ 、 $istart2=istart+len1$ を演算して、ステップS109に進む。ステップS109では、サブルーチンdf-updateを再帰的に呼び出し（ブロックの先頭のistart2とブロック幅のlen2を受け渡す）、サブルーチンを終了する。

【0070】

図27は、サブルーチンbh-updateのフローチャートである。

このサブルーチンは再帰的プログラムである。まず、各スレッドで処理する範囲を示す始点及び終点をis、ieで受ける。また、ブロックEの左上隅の要素がS(k、*)の税所の要素となるように引数で受ける。処理するブロックの先頭istartとブロック幅をlenを受取る。

【0071】

ステップ S 1 1 5 では、 $len < 10$ か否かを判断する。ステップ S 1 1 5 の判断が NO の場合には、ステップ S 1 1 9 に進む。ステップ S 1 1 5 の判断が YES の場合には、ステップ S 1 1 6 において、 $i=1$ と設定し、ステップ S 1 1 7 において、 $i \leq len$ か否かを判断する。ステップ S 1 1 7 の判断が NO の場合には、サブルーチンを抜ける。ステップ S 1 1 7 の判断が YES の場合には、ステップ S 1 1 8 において、 $j=istart+i-1$ 、 $j2=j-nbase$ 、 $je=istart+len-1$ 、 $je2=je-nbase$ 、 $A(is:ie, j) = -A(is:ie, j) \times S(j2, j2)$ 、 $A(is:ie, j) = A(is:ie, j) - A(is:ie, j+1:je) \times S(j2+1:je2, j2)$ を演算して、ステップ S 1 1 7 に進む。

【0072】

ステップ S 1 1 9 では、 $len \geq 32$ あるいは $len \leq 20$ か否かを判断する。ステップ S 1 1 9 の判断が NO の場合には、ステップ S 1 2 1 に進み、判断が YES の場合には、ステップ S 1 2 0 に進む。ステップ S 1 2 0 においては、 $len1=len/2$ 、 $len2=len-len1$ を計算し、ステップ S 1 2 2 に進む。ステップ S 1 2 1 においては、 $len1=len/3$ 、 $len2=len-len1$ を演算し、ステップ S 1 2 2 に進む。ステップ S 1 2 2 においては、サブルーチン `bh-update` を再帰的に呼び出し、対象ブロックの先頭 $istart$ とブロック幅 $len1$ を受け渡す。ステップ S 1 2 3 では、 $js=istart$ 、 $je=istart+len1-1$ 、 $js2=js-nbase$ 、 $je2=js2+len1-1$ 、 $js3=istart+len1$ 、 $je3=js2+len2-1$ 、 $js4=js3-nbase$ 、 $je4=js4+len2-1$ 、 $A(is:ie, js:je) = A(is:ie, js:je) - A(is:ie, js3:je3) \times S(js4:jse, js2:je2)$ 、 $istart2=istart+len1$ を演算する。ステップ S 1 2 4 においては、サブルーチン `bh-update` を再帰的に呼び出し、対象ブロックの先頭 $istart2$ とブロック幅 $len1$ を受け渡して、サブルーチンを抜ける。

【0073】

図 28 は、サブルーチン `exchgrow` のフローチャートである。

ステップ S 1 3 0 において、バリア同期を取る。ステップ S 1 3 1 において、各スレッドで分担する始点、終点を計算する。すなわち、 $len=(nbase+numthrd-1)/numthrd$ 、 $is=(nothrd-1) \times len+1$ 、 $ie=\min(nbase, nothrd \times len)$ 、 $j=nbase+1$ を計算する。ステップ S 1 3 2 では、 $j > \min(n-1, nbase+iblks)$ であるか否かを判

断し、判断がYESの場合には、ステップS136に進み、判断がNOの場合には、ステップS133に進む。ステップS133では、 $ip(j) > j$ か否かを判断する。ステップS133の判断がNOの時は、ステップS135に進む。ステップS133の判断がYESの時は、ステップS134において、 $A(j, is:ie)$ と $A(ip(j), is:ie)$ を入れ替え、ステップS135に進む。ステップS135において、 $j=j+1$ を演算し、ステップS132に戻る。ステップS136では、バリア同期を取って、サブルーチンを抜ける。

【0074】

図29は、サブルーチンexchgcolのフローチャートである。

ステップS140において、バリア同期を取る。ステップS141において、各スレッドで分担する始点、終点を計算する。すなわち、 $len = (n + numthrd - 1) / numthrd$ 、 $is = (nothrd - 1) \times len + 1$ 、 $ie = \min(nbase, nothrd \times len)$ 、 $j = n - 1$ を計算し、ステップS142に進む。ステップS142では、 $j < 1$ を判断する。ステップS142の判断がYESの場合には、ステップS146に進む。ステップS142の判断がNOの場合には、ステップS143において、 $jp(j) > j$ を判断する。ステップS143の判断がNOの場合には、ステップS145に進む。ステップS143の判断がYESの場合には、ステップS144において、 $A(is:ie, j)$ と $A(is:ie, jp(j))$ を入れ替えて、ステップS145に進む。ステップS145では、 $J=J-1$ として、ステップS142に戻る。ステップS146では、バリア同期を取って、サブルーチンを抜ける。

行列計算の一般的アルゴリズムに関しては、以下の教科書を参照されたい。

【0075】

G. H. Golub and C. F. Van Loan "Matrix Computations" The Johns Hopkins University Press, Third edition 1996

(付記1) 共有メモリ型スカラ並列計算機用逆行列の並列処理方法であって、

逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、

該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右横、右下のそれぞれのブロックに分解するステップと、

該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列にLU分解するステップと、

左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、

所定の正方形ブロックの更新を複数段に分けて、1つのプロセッサで行うステップと、

該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと、
を備える方法。

【0076】

(付記2) 前記共有メモリ型スカラ並列計算機は、複数のプロセッサと、該プロセッサに対応して設けられる複数のキャッシュメモリと、複数の共有メモリと、これらを通信可能なように接続する相互結合網とからなることを特徴とする付記1に記載の方法。

【0077】

(付記3) 上記方法は、G a u s s - J o r d a nの方法をブロック毎に並列に計算する構成としたことを特徴とする付記1に記載の方法。

(付記4) 前記各ブロックを分割して並列計算するさいの分割の幅は、逆行列を求める行列の大きさと並列処理のために利用できるプロセッサ数から、並列処理しない正方形ブロックの演算量の総和が演算全体の1%程度となるように設定されることを特徴とする付記1に記載の方法。

【0078】

(付記5) 共有メモリ型スカラ並列計算機用逆行列の並列処理方法を実現させるプログラムであって、

逆行列を求めるべき行列内に、所定の正方形ブロックを指定するステップと、
該行列を該正方形ブロックを中心として左上、左横、左下、上、下、右上、右



横、右下のそれぞれのブロックに分解するステップと、

該分解されたそれぞれのブロックをプロセッサの数に応じて分割し、該正方形ブロックと、この下、右横、右下のブロックを並列にLU分解するステップと、

左横、上、下、右横のブロックを再帰的プログラムで並列に更新し、左上、左下、右上、右下のブロックを該再帰的プログラムで更新されたブロックを用いて並列に更新を行うステップと、

所定の正方形ブロックの更新を複数段に分けて、1つのプロセッサで行うステップと、

該正方形ブロックの位置を順次、該行列の対角線上を移動するように設定し、上記ステップを繰り返すことにより該行列の逆行列を求めるステップと、を備える方法を共有メモリ型スカラ並列計算機に実行させるプログラム。

【0079】

(付記6) 前記共有メモリ型スカラ並列計算機は、複数のプロセッサと、該プロセッサに対応して設けられる複数のキャッシュメモリと、複数の共有メモリと、これらを通信可能なように接続する相互結合網とからなることを特徴とする付記5に記載のプログラム。

【0080】

(付記7) 上記方法は、G a u s s - J o r d a nの方法をブロック毎に並列に計算する構成としたことを特徴とする付記5に記載のプログラム。

(付記8) 前記各ブロックを分割して並列計算するさいの分割の幅は、逆行列を求める行列の大きさと並列処理のために利用できるプロセッサ数から、並列処理しない正方形ブロックの演算量の総和が演算全体の1%程度となるように設定されることを特徴とする付記5に記載のプログラム。

【0081】


【発明の効果】

高性能かつスケーラビリティのある逆行列の解法を実現できる。

【図面の簡単な説明】

【図1】

本発明の実施形態が前提とする共有メモリ型スカラ計算機のハードウェア構成



を示した図である。

【図 2】

本発明の実施形態の計算順序を説明する図（その 1）である。

【図 3】

本発明の実施形態の計算順序を説明する図（その 2）である。

【図 4】

本発明の実施形態の計算順序を説明する図（その 3）である。

【図 5】

本発明の実施形態の計算順序を説明する図（その 4）である。

【図 6】

本発明の実施形態の計算順序を説明する図（その 5）である。

【図 7】

本発明の実施形態の計算順序を説明する図（その 6）である。

【図 8】

本発明の実施形態の計算順序を説明する図（その 7）である。

【図 9】

本発明の実施形態の疑似コード（その 1）である。

【図 10】

本発明の実施形態の疑似コード（その 2）である。

【図 11】

本発明の実施形態の疑似コード（その 3）である。

【図 12】

本発明の実施形態の疑似コード（その 4）である。

【図 13】

本発明の実施形態の疑似コード（その 5）である。

【図 14】

本発明の実施形態の疑似コード（その 6）である。

【図 15】

本発明の実施形態の疑似コード（その 7）である。

【図 1 6】

疑似コードの処理をフローチャートで表した図（その 1）である。

【図 1 7】

疑似コードの処理をフローチャートで表した図（その 2）である。

【図 1 8】

疑似コードの処理をフローチャートで表した図（その 3）である。

【図 1 9】

疑似コードの処理をフローチャートで表した図（その 4）である。

【図 2 0】

疑似コードの処理をフローチャートで表した図（その 5）である。

【図 2 1】

疑似コードの処理をフローチャートで表した図（その 6）である。

【図 2 2】

疑似コードの処理をフローチャートで表した図（その 7）である。

【図 2 3】

疑似コードの処理をフローチャートで表した図（その 8）である。

【図 2 4】

疑似コードの処理をフローチャートで表した図（その 9）である。

【図 2 5】

疑似コードの処理をフローチャートで表した図（その 1 0）である。

【図 2 6】

疑似コードの処理をフローチャートで表した図（その 1 1）である。

【図 2 7】

疑似コードの処理をフローチャートで表した図（その 1 2）である。

【図 2 8】

疑似コードの処理をフローチャートで表した図（その 1 3）である。

【図 2 9】

疑似コードの処理をフローチャートで表した図（その 1 4）である。

【符号の説明】

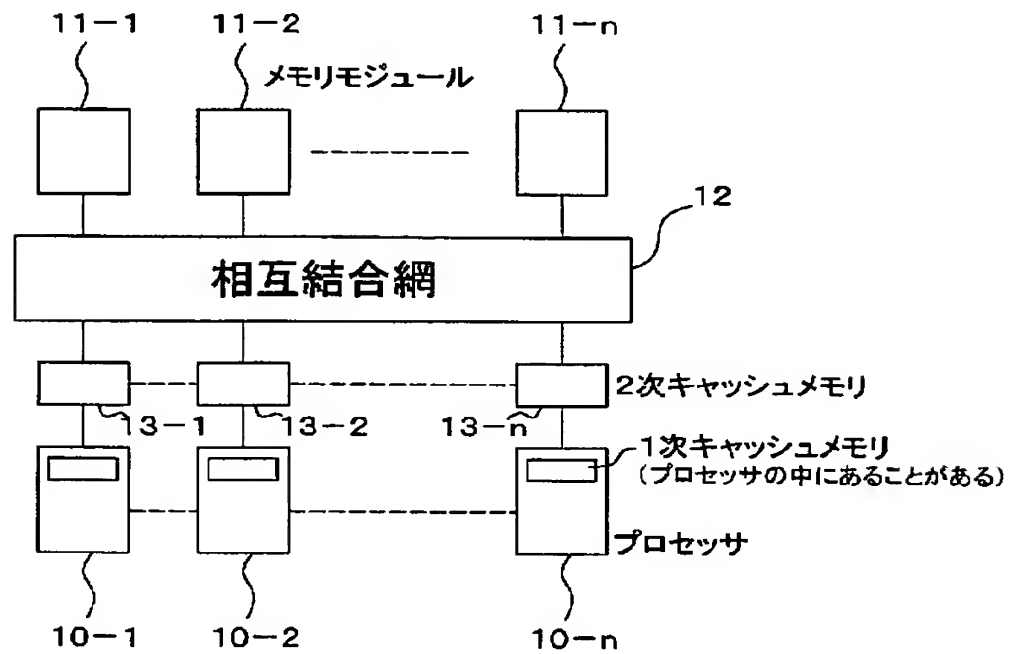


- 1 0 - 1 ~ 1 0 - n プロセッサ
- 1 1 - 1 ~ 1 1 - n メモリモジュール
- 1 2 相互結合網
- 1 3 - 1 ~ 1 3 - n 2 次キャッシュメモリ

【書類名】 図面

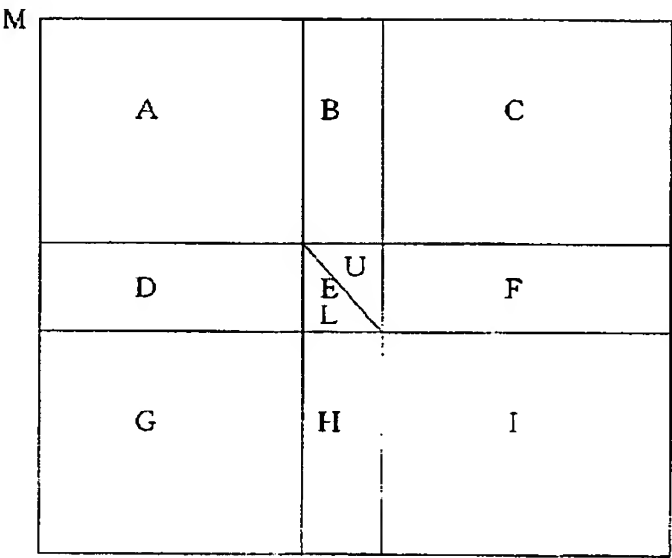
【図 1】

本発明の実施形態が前提とする共有メモリ型
スカラ計算機のハードウェア構成を示した図



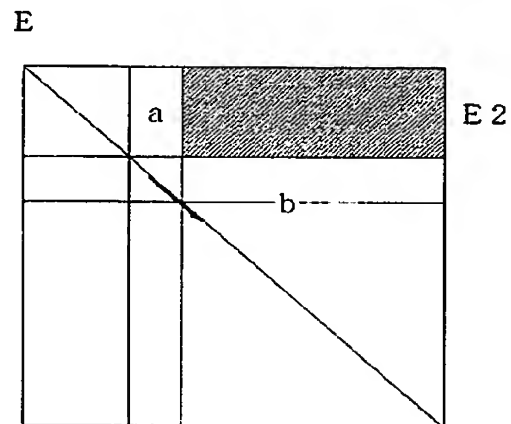
【図 2】

本発明の実施形態の計算順序を説明する図（その 1）



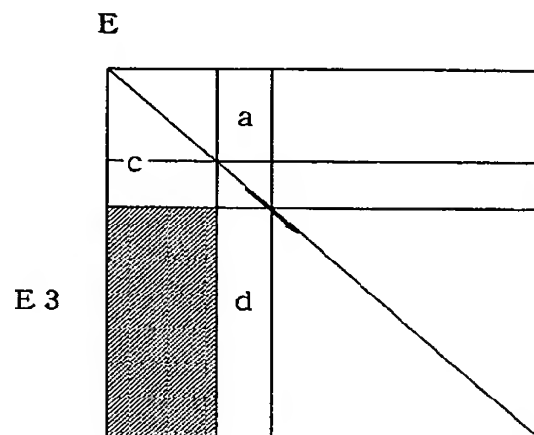
【図 3】

本発明の実施形態の計算順序を説明する図（その 2）



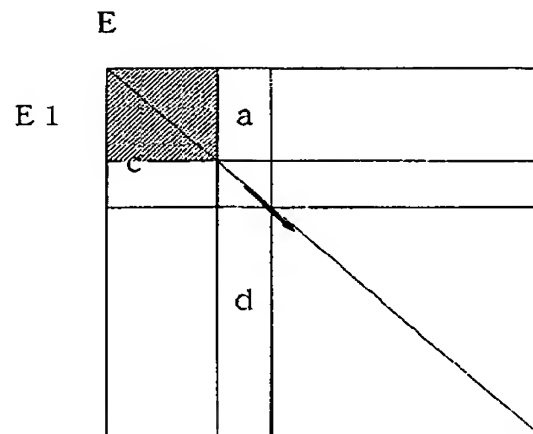
【図 4】

本発明の実施形態の計算順序を説明する図（その 3）



【図 5】

本発明の実施形態の計算順序を説明する図（その 4）



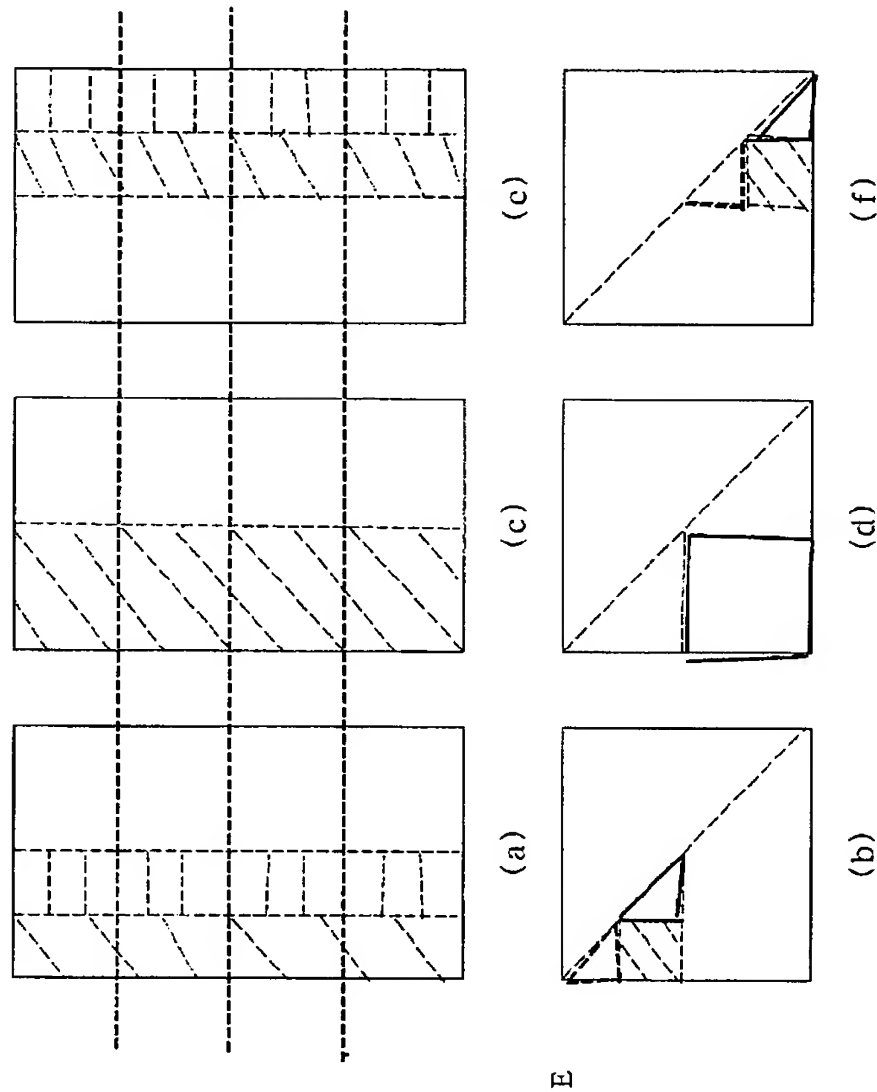
【図 6】

本発明の実施形態の計算順序を説明する図（その 5）

A	B	C
D	E	F
G	H	I

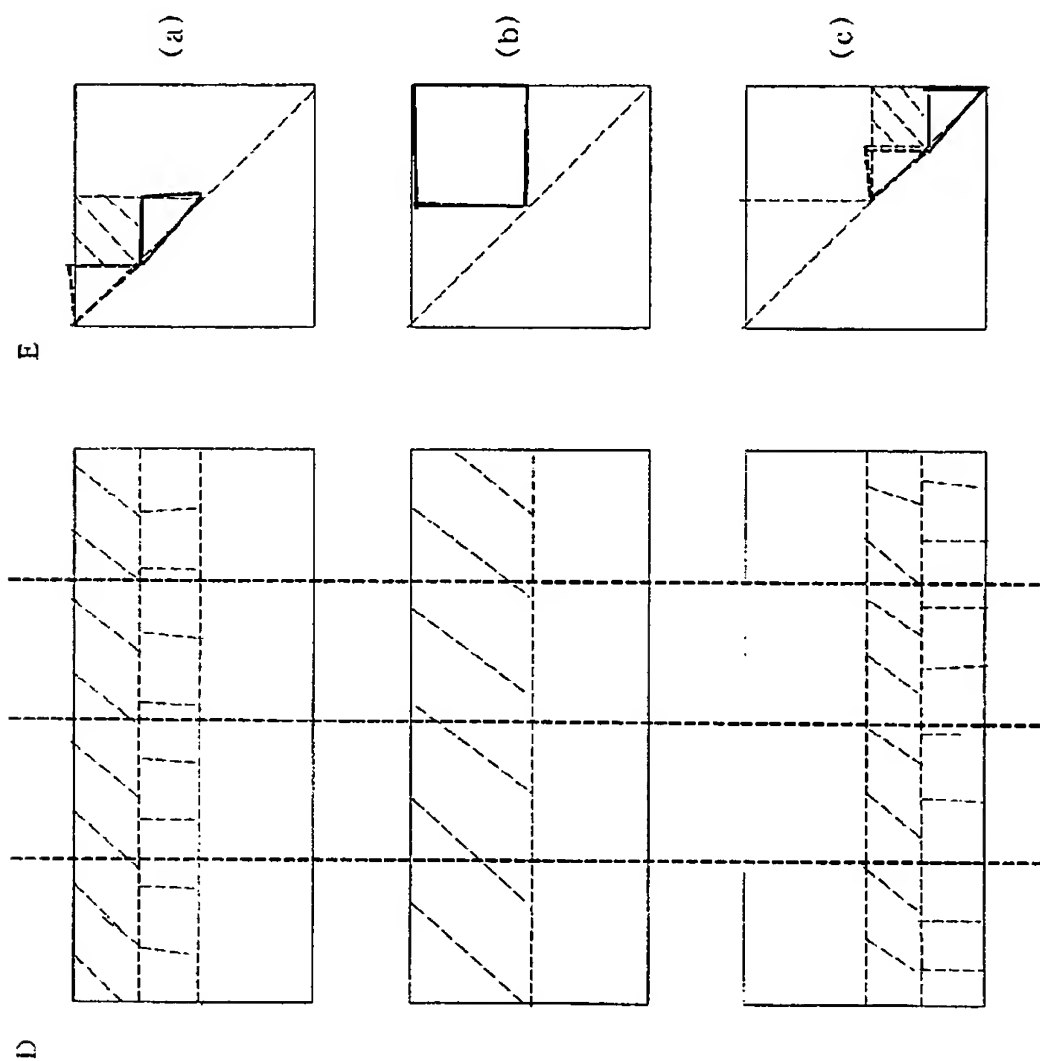
【図 7】

本発明の実施形態の計算順序を説明する図（その 6）



【図 8】

本発明の実施形態の計算順序を説明する図（その 7）



【図 9】

本発明の実施形態の疑似コード（その 1）

```

c   逆行列の並列アルゴリズム
    subroutine inversion(a,k,n)
    shared array a(k,n),ip(n)
    create threads
    set nothrd and numthrd
c   nothrd はスレッドの番号 1 ~ #TH、numthrd=#TH（スレッドの総数）
    iblk=ブロック幅
    nb=(n+blk-1)/blk
    do i=1,nb-1
    nbase=(nb-1)*blk
    call LU(a,k,n,nbase,blk,ip,nothrd,numthrd)
        ! 指定されたブロックの LU 分解を行い行ブロックを更新し、右下の正方
        ! 行列を更新する演算を並列に行う。ip(nbase+1:nbase-bk)に
        ! 行の入れ換えの情報が返却される。この部分の並列アルゴリズムは
        ! 特願平 12-358232 号に記載されているのでそれを利用。
    do i=nbase+1,nbase+blk
    if(ip(i)>i)then
    exchange(a(i,1:nbase),a(ip(i),1:nbase))
    endif
    enddo
    call update(a,k,n,nbase,blk,nothrd,numthrd)
    enddo
    nbase=(nb-1)*blk
    blklast=n-nbase
    call LU(a,k,n,nbase,blklast,nothrd,numthrd)
    call update(a,k,n,nbase,blklast,nothrd,numthrd)
    return
    end

```

【図 10】

本発明の実施形態の疑似コード（その 2）

```

c   ブロック LU 分解の情報を利用して残りの部分の更新を行うルーチン
    subroutine update(a,k,n,nbase,blk,nothrd,numthrd)
    call b-update(a,k,n,nbase,blk,nothrd,numthrd)
    call d-update(a,k,n,nbase,blk,nothrd,numthrd)
    call c-update(a,k,n,nbase,blk,nothrd,numthrd)
    call a-update(a,k,n,nbase,blk,nothrd,numthrd)
    call g-update(a,k,n,nbase,blk,nothrd,numthrd)
    BARRIER SYNC
    if(nothread=1) then
c   e の更新 1
    call e-update1(a(nbase+1,nbase+1),k,n,blk)
    endif
    BARRIER SYNC
    len←(nbase+numthrd-1)/numthrd
    is←(nothrd-1)*len+1
    ie←nothrd*len
    call df-update(a(nbase+1,1),k,n,a(nbase-1,nbase-1),is,ie,blk)
    nbase2←nbase-bk
    len←(n-nbase2-numthrd-1)/numthrd
    is2←nbase2-(nothrd-1)*len+1
    ie2←nbase2-nothrd*len
    call df-update(a(nbase+1,1),k,n,a(nbase+1,nbase-1),is,ie,blk)
    BARRIER SYNC
    if(nothread=1) then
c   e の更新 2
    call e-update2(a(nbase+1,nbase+1),k,n,blk)
    endif
    BARRIER SYNC
    len←(nbase+numthrd-1)/numthrd
    is←(nothrd-1)*len+1
    ie←nothrd*len
    call bh-update(a(1,nbase+1),k,n,a(nbase+1,nbase+1),is,ie,blk)
    nbase2←nbase+blk
    len←(n-nbase2+numthrd-1)/numthrd
    is2←nbase2+(nothrd-1)*len-1
    ie2←nbase2+nothrd*len
    call bh-update(a(1,nbase+1),k,n,a(nbase-1,nbase+1),is,ie,blk)
    BARRIER SYNC
    if(nothread=1) then
c   e の更新 3
    call e-update3(a(nbase+1,nbase+1),k,n,blk)
    endif
    BARRIER SYNC
    len←(n+numthrd-1)/numthrd
    is←(nothrd-1)*len+1
    ie←min(n,nothrd*len)
    if(ip(i)>i) then
    exchange(a(is:ie,i),a(is,ie,ip(i)))
    endif
    enddo
    BARRIER SYNC

    eliminate threads

    return
    end

```

【図 11】

本発明の実施形態の疑似コード（その 3）

```

c   ブロック B の更新
    subroutine b-update(a,k,n,nbase,blk,nothrd,numthrd)
    shared array a(k,n)
    len←(nbase+numthrd-1)/numthrd
    isl←(nothrd-1)*len
    iel←nothrd*len
    icf=nbase
    a(is:ie,i of+1:i of+blk)
    ← a(is:ie,i of-1:i of+blk)*TRU-U(i of+1:i of+blk,i of-1:i of+blk)-1
    ! TRU-U 対角要素を 1.0 にした上三角行列
    return
    end

c   ブロック C の更新
    subroutine d-update(a,k,n,nbase,blk,nothrd,numthrd)
    shared array a(k,n)
    len←(nbase+numthrd-1)/numthrd
    isl←(nothrd-1)*len
    iel←nothrd*lenisl
    i of=nbase
    a(is:ie,i of+1:i of+blk)
    ←TRL(i of+1:i of+blk,i of-1:i of+blk)-1*a(is:ie,i of-1:i of-1+blk)
    ! TRL は下三角行列
    return
    end

```

【図 12】

本発明の実施形態の疑似コード(その4)

```
c   ブロック C の更新
    subroutine c-update(a,k,n,nbase,blk,nothrd,numthrd)
    shared array a(k,n)
    nbase2←nbase+blk
    iof=nbase
    len←(n-nbase2+numthrd-1)/numthrd
    is2←nbase2+(nothrd-1)*len+1
    ie2←nbase2+nothrd*len
    a(1:iof,is2:ie2)
    ←a(1:iof,is2:ie2)
    -a(1:iof,iof+1:iof+blk)*a(iof+1:iof+blk,is2:ie2)
    return
    end

c   ブロック A の更新
    subroutine a-update(a,k,n,nbase,blk,nothrd,numthrd)
    shared array a(k,n)
    len←(nbase+numthrd-1)/numthrd
    is2←(nothrd-1)*len
    ie2←nothrd*len
    iof=nbase
    a(1:iof,is2:ie2)
    ←a(1:iof,is2:ie2)
    -a(1:iof,iof+1:iof+blk)*a(iof+1:iof+blk,is2:ie2)
    return
    end
```

【図 13】

本発明の実施形態の疑似コード(その5)

```

c   ブロック G の更新
    subroutine g-update(a,k,n,nbase,blk,nothrd,numthrd)
    shared array a(k,n)
    len←(nbase+numthrd-1)/numthrd
    is2←(nothrd-1)*len
    ie2←nothrd*len
    iof←nbase+blk
    a(iof+1:n,is2:ie2)
    ←a(iof+1:n,is2:ie2)
    -a(iof+1:n,iof+1:iof+blk)*a(iof+1:iof+blk,is2:ie2)
    return
    end

c   ブロック E の最初の更新
    subroutine e-update1(s,k,n,blk)
    shared array s(k,n)

    do i=1,blk
    s(1:i-1,i+1:blk)←s(1:i-1,i+1:blk)-s(1:i-1,i)*s(i,i+1:blk)
    enddo
    return
    end

c   ブロック E の2回目の更新
    subroutine e-update2(s,k,n,blk)
    shared array s(k,n)

    do i=1,blk
    tmp←1.0/s(i,i)
    s(i,1:i-1)←tmp*s(i,1:i-1)
    s(i+1:blk,1:i-1)←s(i+1:blk,1:i-1)-s(i,1:i-1)*s(i+1:blk,i)
    s(i+1:blk,i)←-s(i+1:blk,i)*tmp
    s(i,i)←tmp
    enddo
    return
    end

```

【図 14】

本発明の実施形態の疑似コード（その6）

```

c   ブロック E の最後の更新
    subroutine e-update3(s,k,n,blk)
    shared array s(k,n)

    do i=1,blk
    s(1:i-1,1:i-1)←s(1:i-1,1:i-1)-s(1:i-1,i)*s(i,1:i-1)
    s(1:i-1,i)←-s(1:i-1,i)*a(i,i)
    enddo
    return
    end

c   ブロック D および F の更新
    subroutine df-update(a,k,n,s,is,ie,len)
    shared array a(k,*),s(k,*)
    if(len<10)then
    do i=1,len
    a(1:i-1,is:ie)←a(1:i-1,is:ie)-s(1:i-1,i)*a(i,is:ie)
    enddo
    else
    if(len>=32 or len<=20)then
    len1←len/2
    len2←len-len1
    else
    len1←len/3
    len2←len-len1
    endif
    call df-update(a,k,n,s,is,ie,len1)
    a(1:len1,is:ie)
    ←a(1:len1,is:ie)-s(1:len1:len1+1:len)*a(len1+1:len,is,ie)
    call df-update(a(len1+1,1),k,n,s(len1+1,len1+1),is,ie,len2)
    endif
    return
    end

```

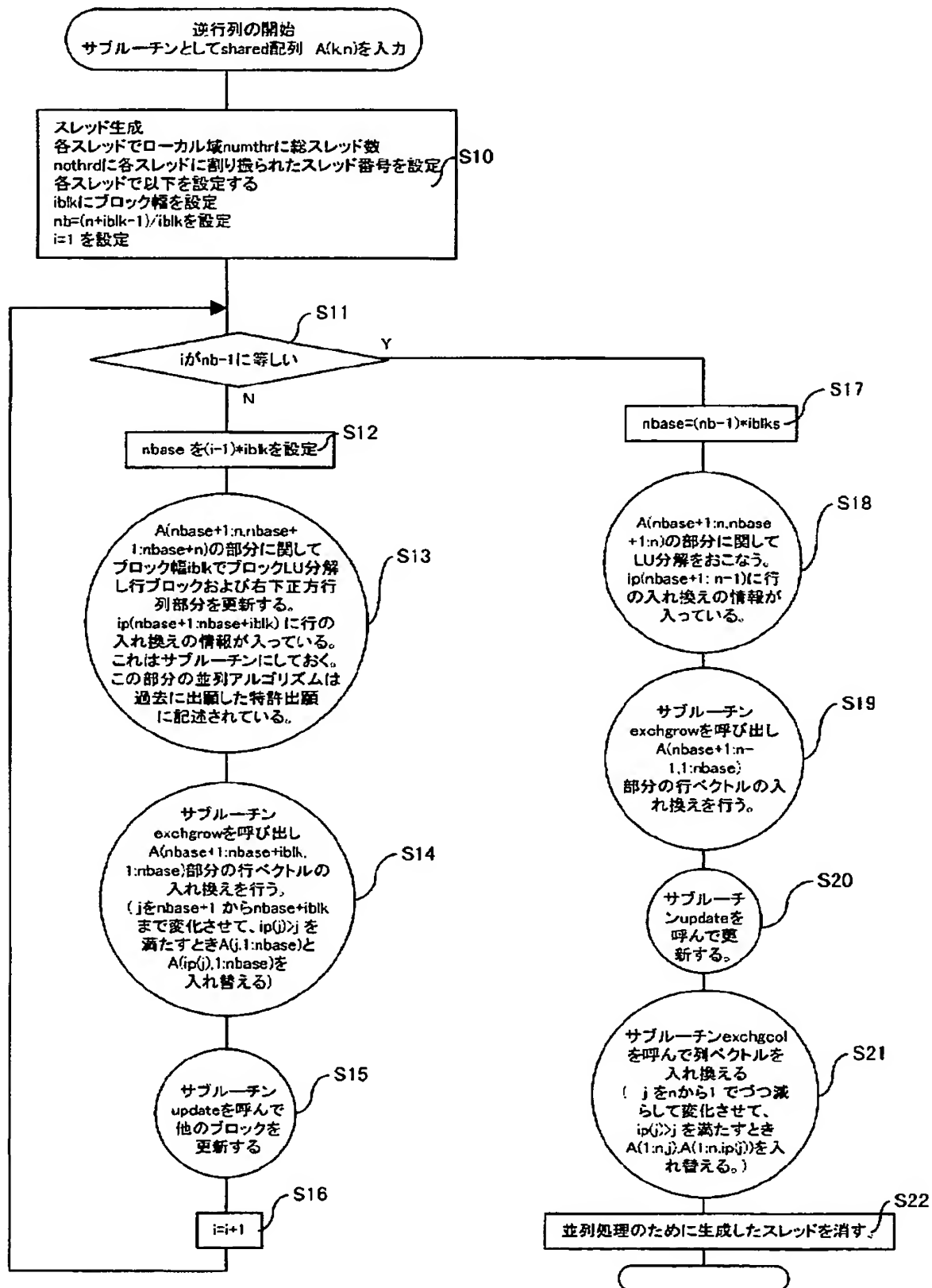
【図 15】

本発明の実施形態の疑似コード（その 7）

```
c   ブロック B および H の更新
    dubroutine bh-update(a,k,n,s,is,ie,len)
    shared array a(k,*),s(k,*)
    if(len<10) then
    do i=1,len
    a(is:ie,i)←-a(is:ie,i)*s(i,i)
    a(is:ie,i)←a(is:ie,i)-a(is:ie,i+1:len)*s(i+1:len,i)
    enddo
    else
    if(len>=32 or len<=20) then
    len1←len/2
    len2←len-len1
    else
    len1←len/3
    len2←len-len1
    endif
    call bh-update(a,k,n,s,is,ie,len1)
    a(is:ie,1:len1)←a(is:ie,1:len1)
    -a(is:ie,len1+1:len)*s(len1+1:len,1:len1)
    call bh-update(a(1,len1+1),k,n,s(len1+1,len1+1),is,ie,len2)
    endif
    return
    end
```

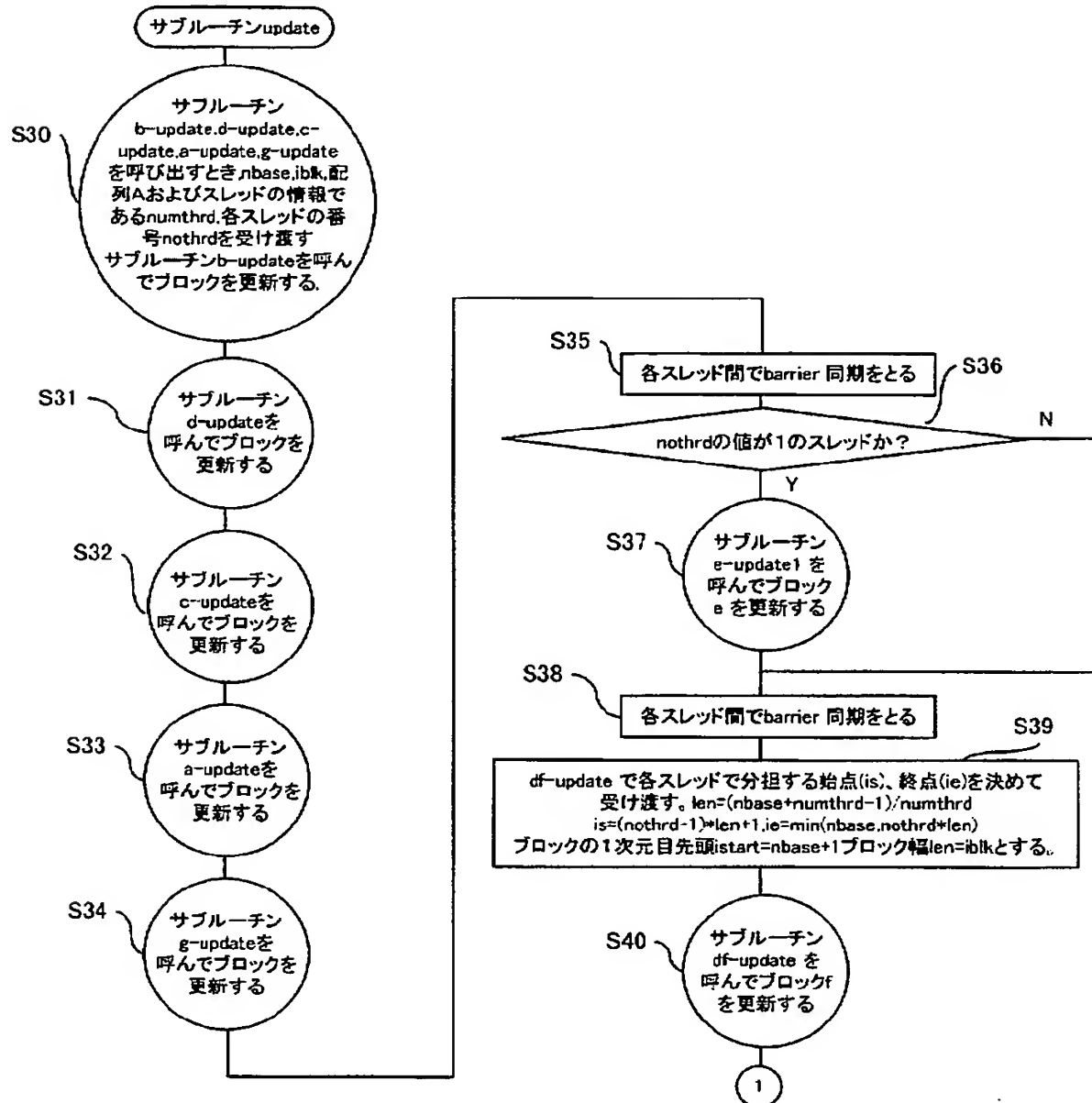
【図 16】

疑似コードの処理をフローチャートで表した図(その1)



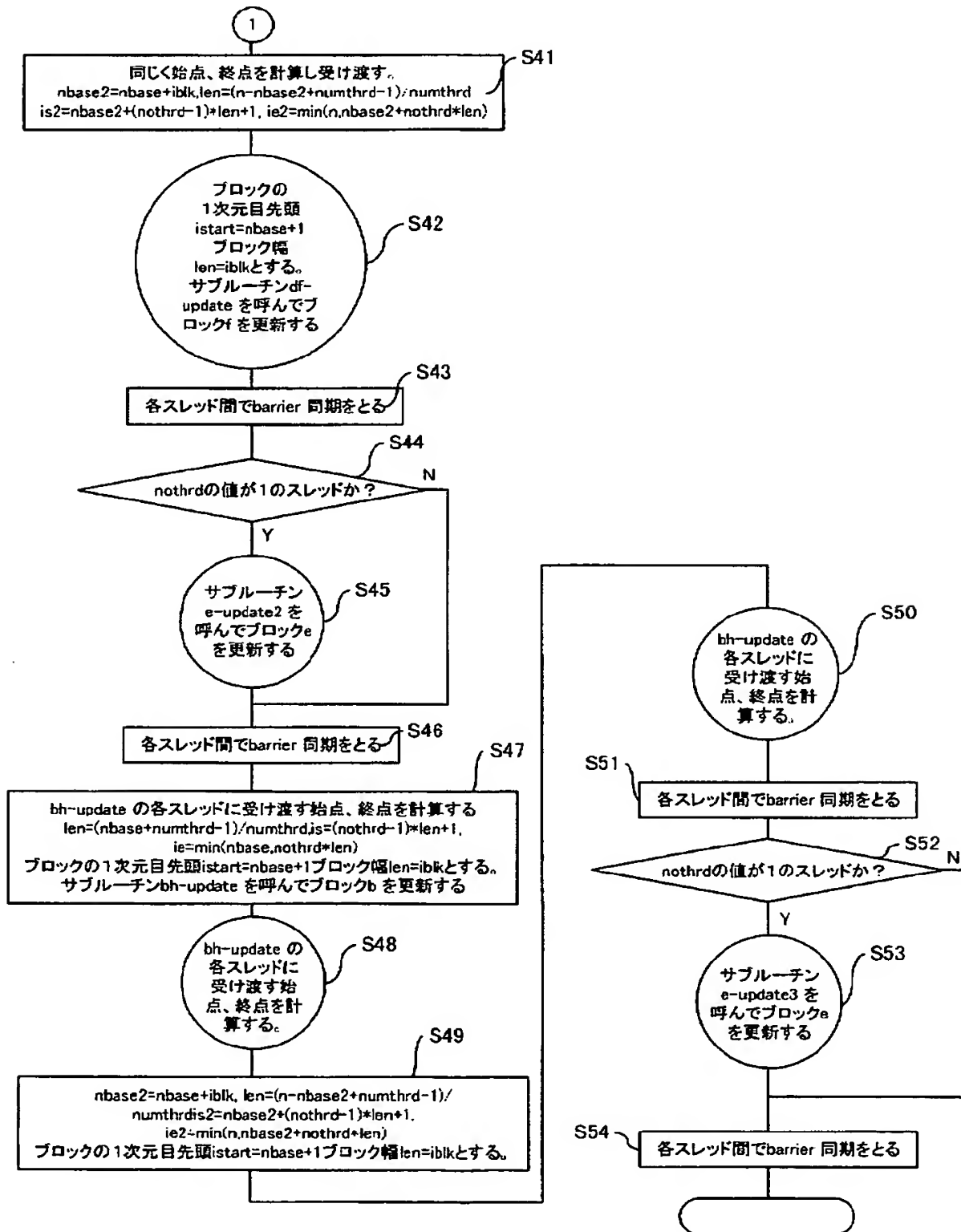
【図 17】

疑似コードの処理をフローチャートで表した図(その2)



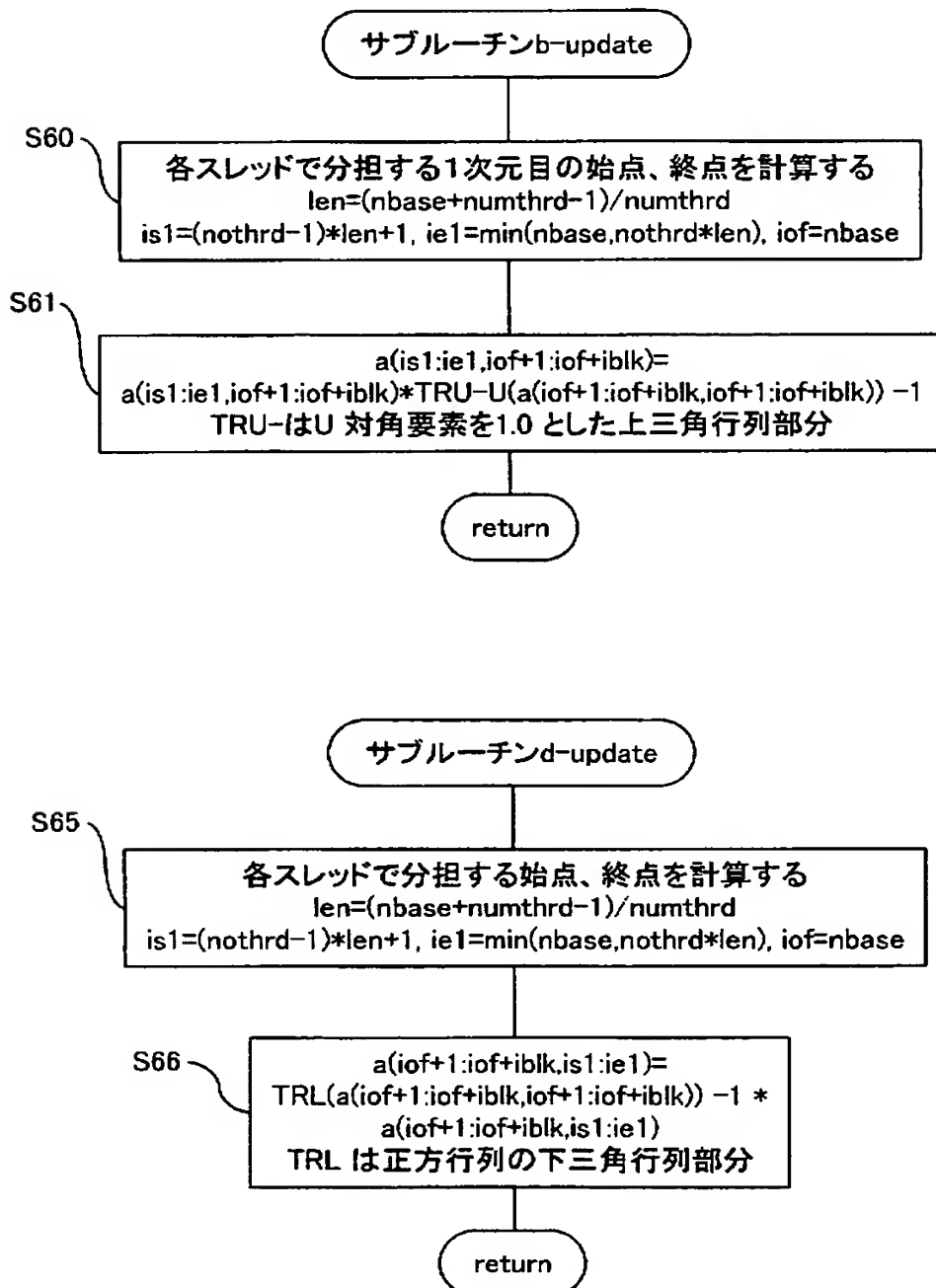
【図 18】

疑似コードの処理をフローチャートで表した図(その3)



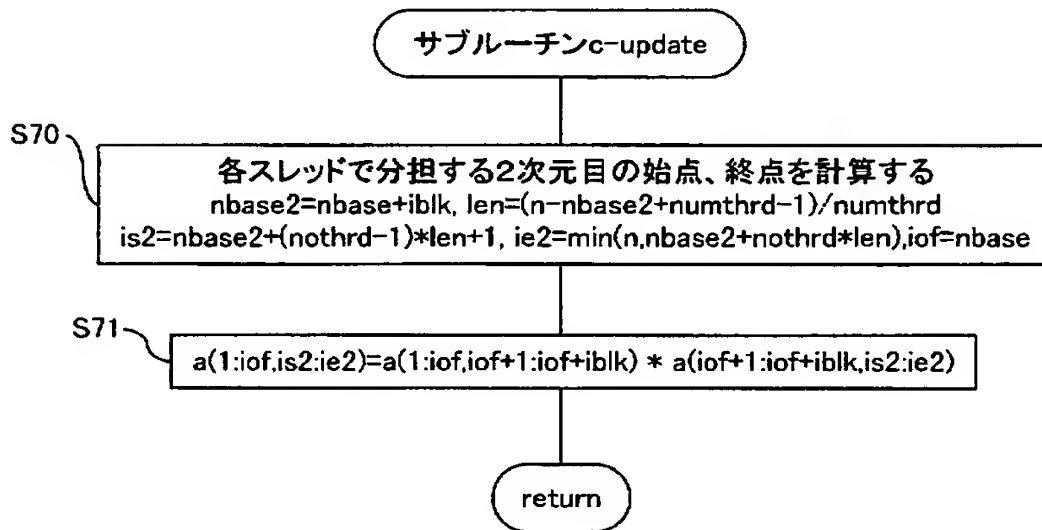
【図19】

疑似コードの処理をフローチャートで表した図(その4)



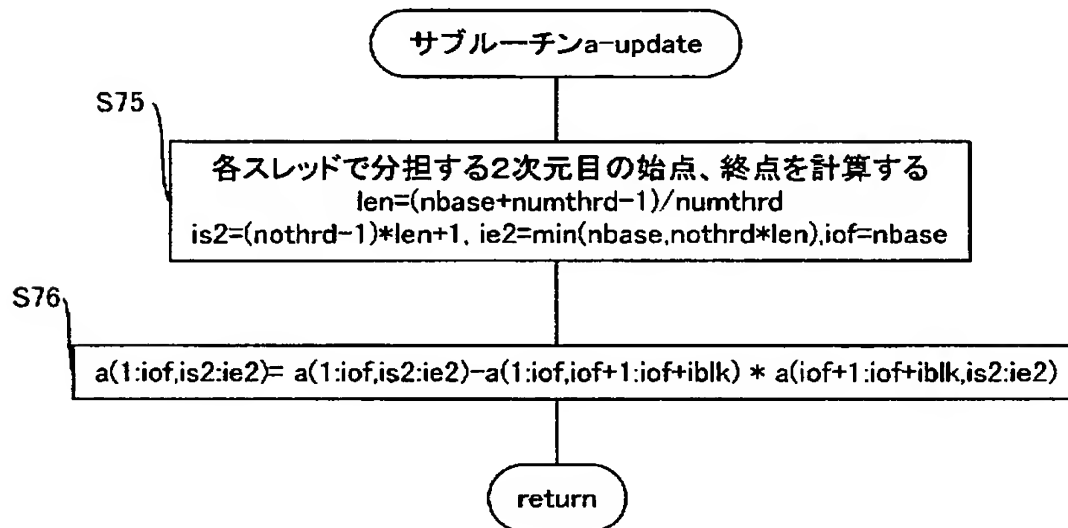
【図 20】

疑似コードの処理をフローチャートで表した図(その5)



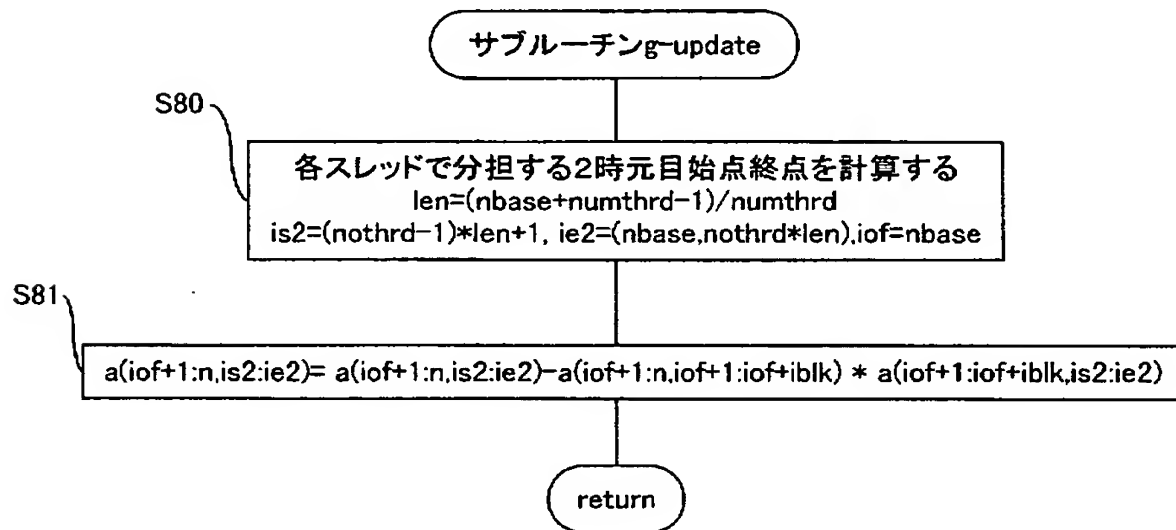
【図 21】

疑似コードの処理をフローチャートで表した図(その6)



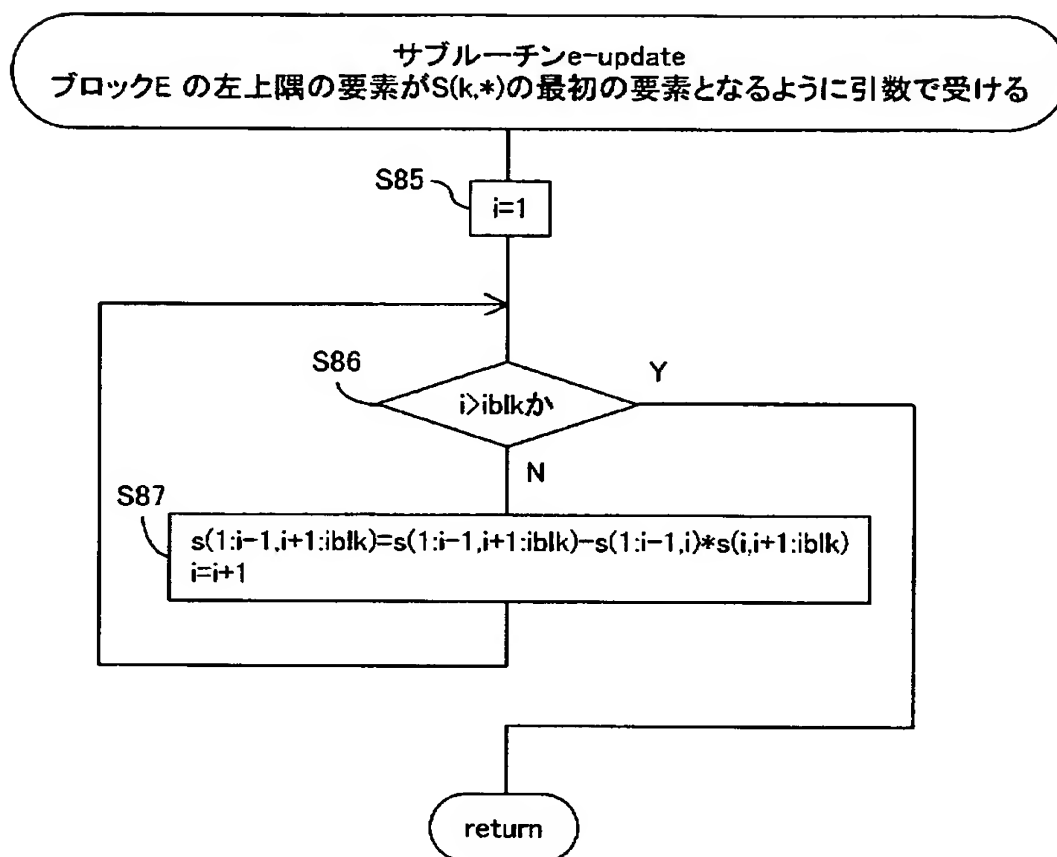
【図 22】

疑似コードの処理をフローチャートで表した図(その7)



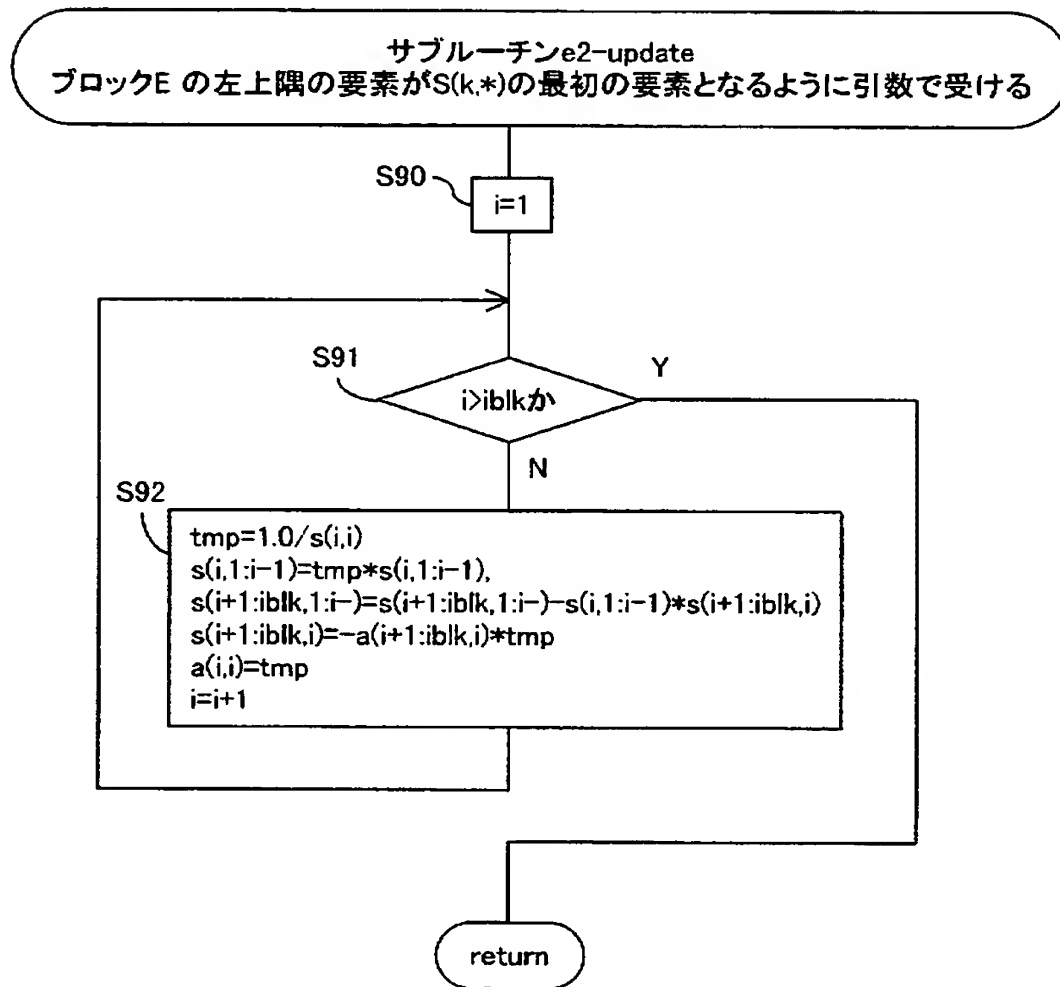
【図 23】

疑似コードの処理をフローチャートで表した図(その8)



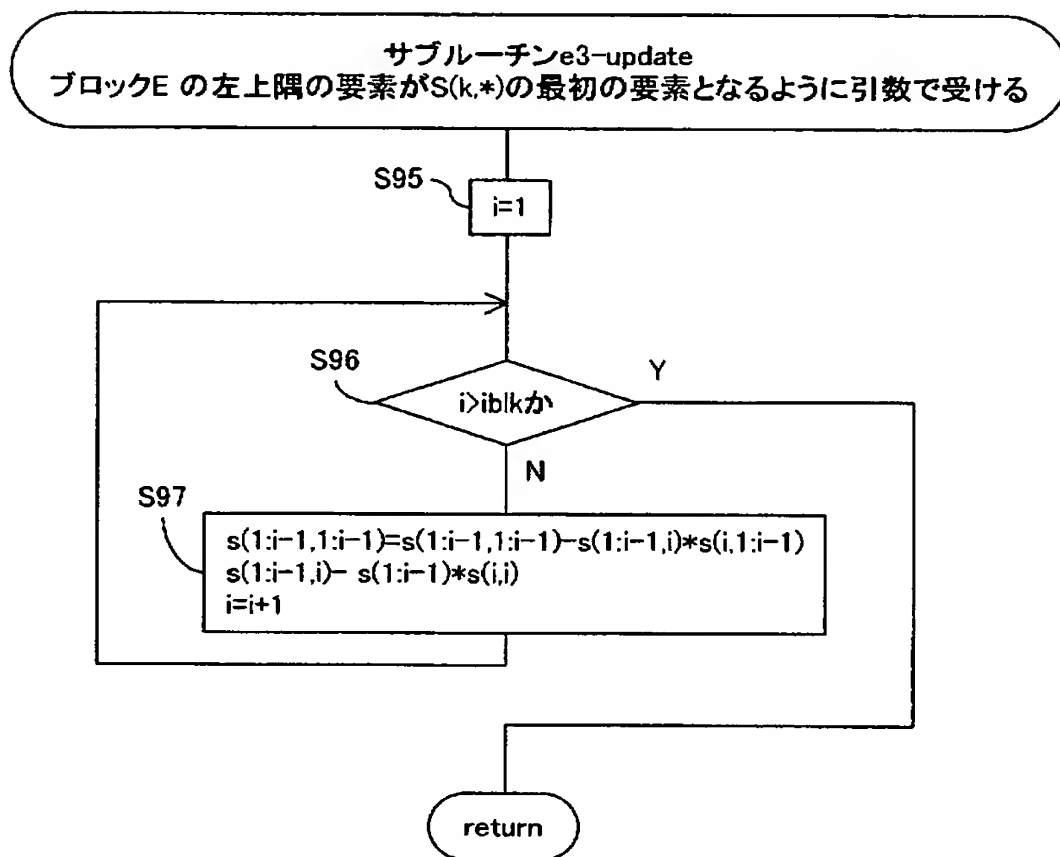
【図 24】

疑似コードの処理をフローチャートで表した図(その9)



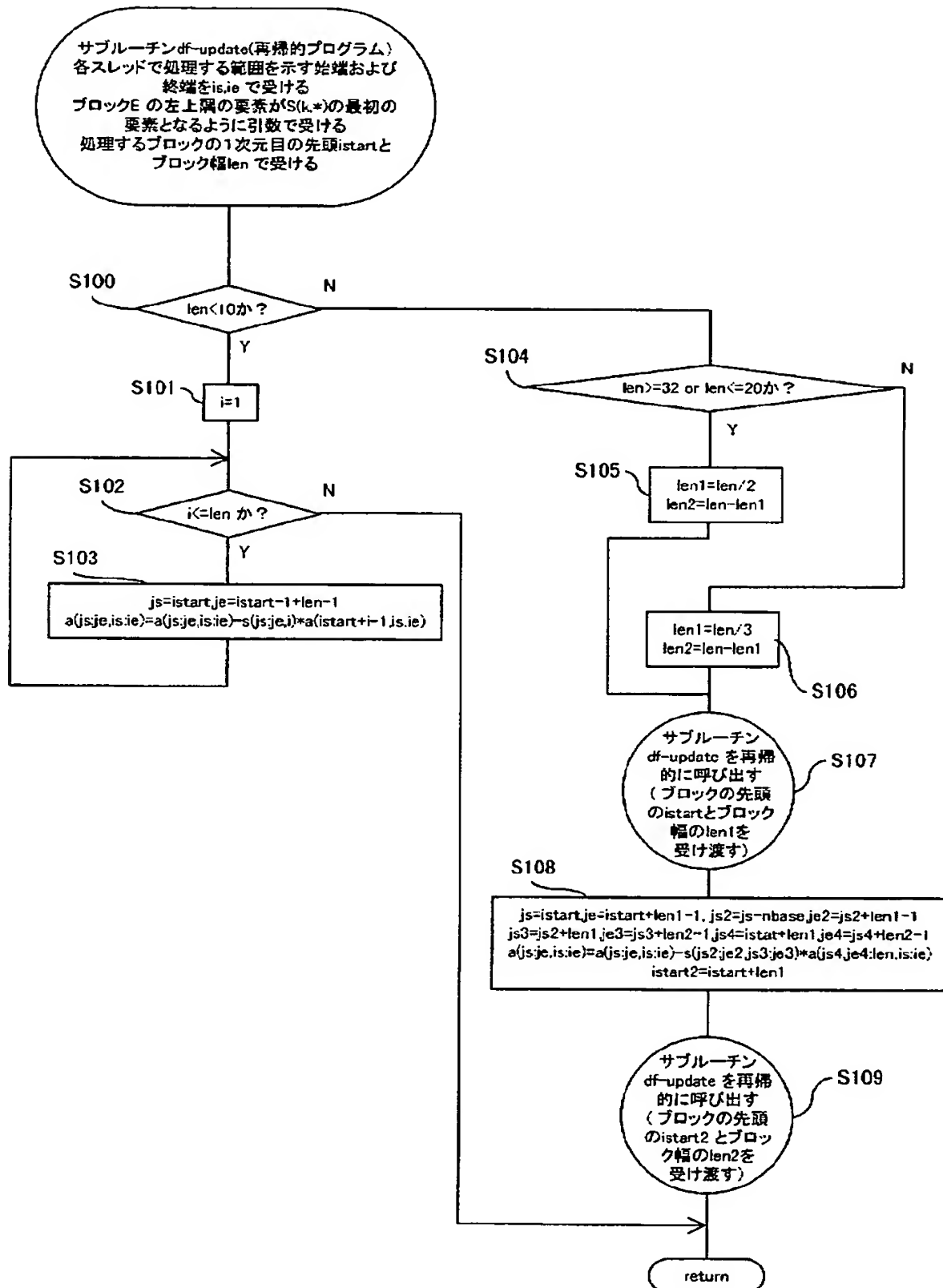
【図 25】

疑似コードの処理をフローチャートで表した図(その10)



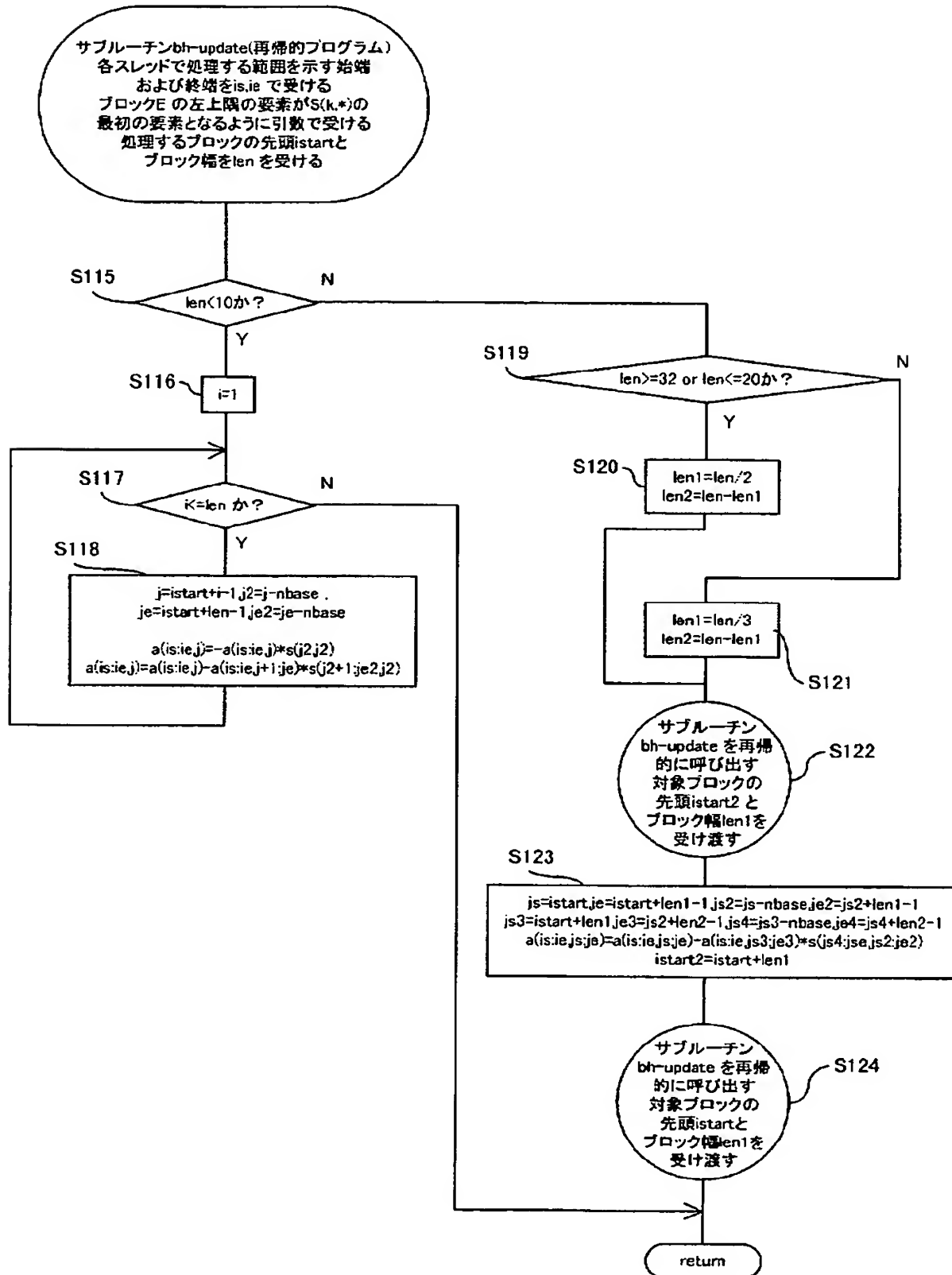
【図 26】

疑似コードの処理をフローチャートで表した図(その11)



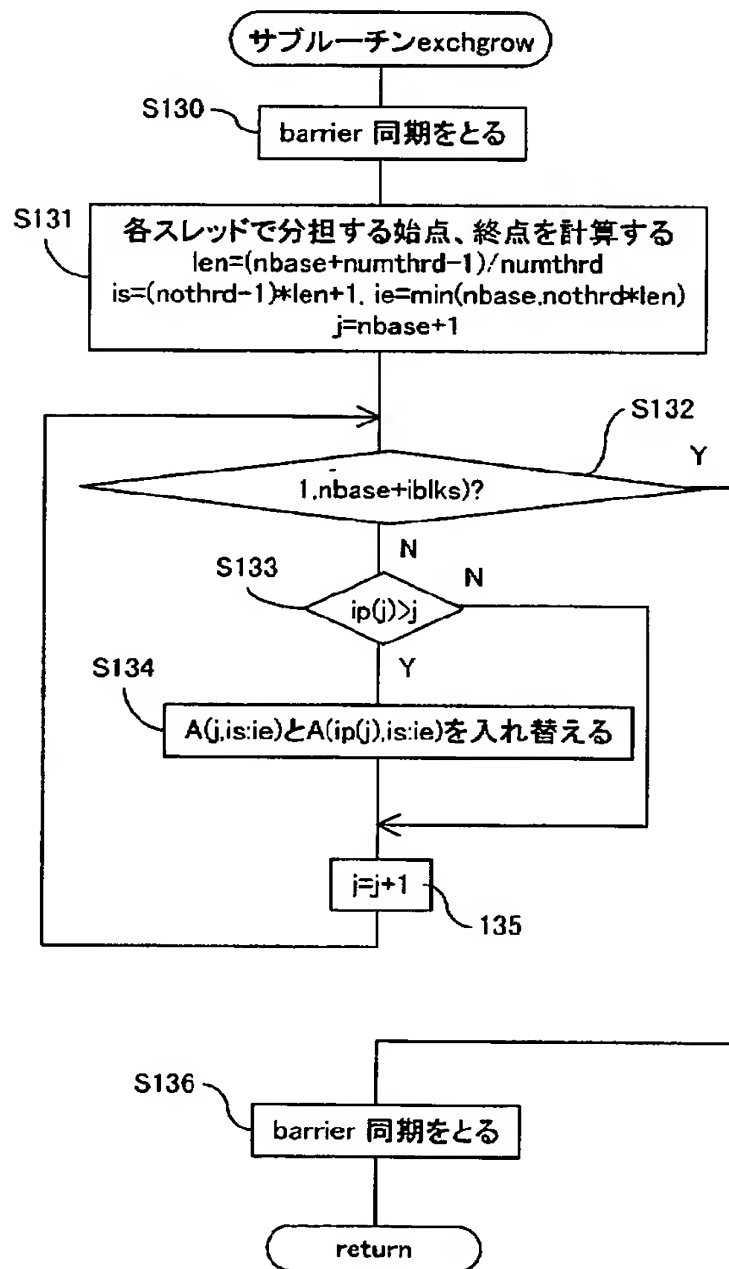
【図 27】

疑似コードの処理をフローチャートで表した図(その12)



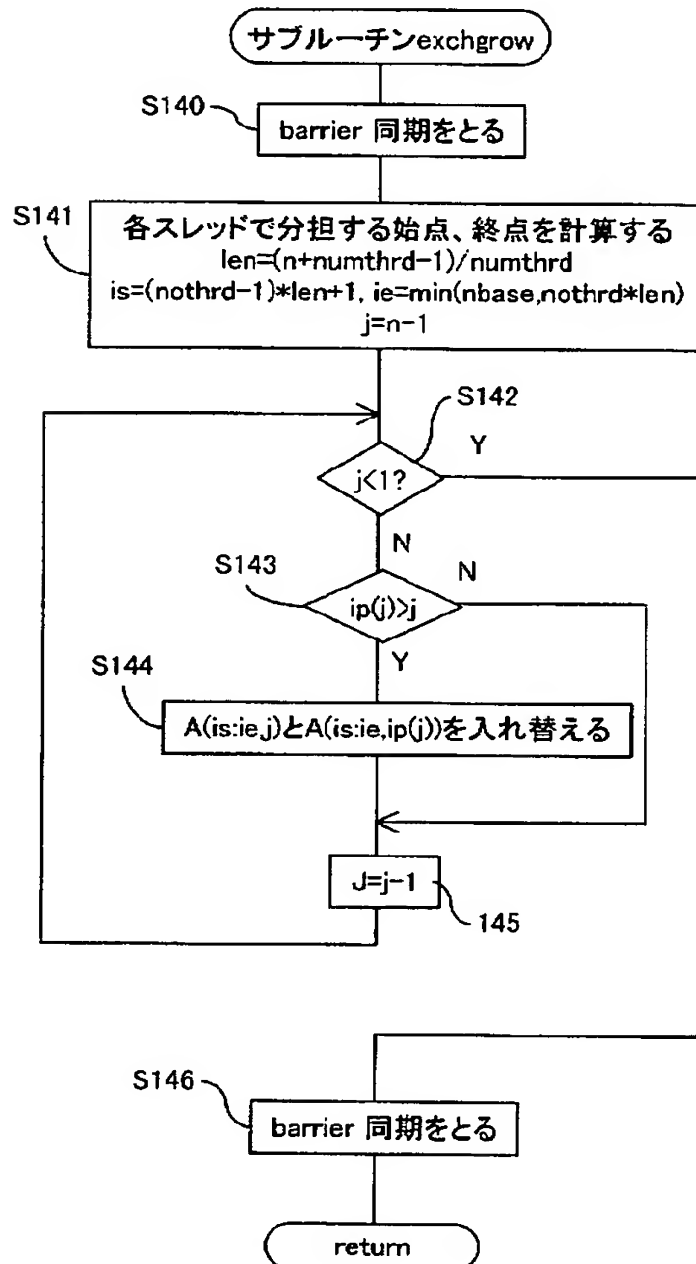
【図 28】

疑似コードの処理をフローチャートで表した図(その13)



【図 29】

疑似コードの処理をフローチャートで表した図(その14)



【書類名】 要約書

【要約】

【課題】 共有メモリ型スカラ計算機において、高速に逆行列の演算が行える演算の並列処理方法を提供する。

【解決手段】 まず、ブロック E と H L U 分解する。次に、ブロック B をブロック E の上三角部分を使って更新し、ブロック E の下三角部分を使って、ブロック D を更新する。このとき、L U 分解によって、ブロック F と I が更新済みである。そして、ブロック B、D、F、H を使って、ブロック A、C、G、I を更新し、ブロック E の上三角部分を更新し、最後にブロック D、F を更新する。そして、ブロック E を 2 回目の更新をして、その結果を用いて、ブロック B、H を更新する。最後に、ブロック E の最終的な更新をして、ピボットの入れ替え処理を終えて処理を終了する。これらのブロックの処理は、複数のスレッドに分割され、平行して行われる。

【選択図】 図 2

特願 2003-074548

出願人履歴情報

識別番号

[000005223]

1. 変更年月日 1990年 8月24日
[変更理由] 新規登録
住 所 神奈川県川崎市中原区上小田中1015番地
氏 名 富士通株式会社
2. 変更年月日 1996年 3月26日
[変更理由] 住所変更
住 所 神奈川県川崎市中原区上小田中4丁目1番1号
氏 名 富士通株式会社